DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

TRATAMIENTO DE IMÁGENES, EXTRACCIÓN DE CARACTERÍSTICAS Y "*MATCHING*" EN PLATAFORMAS ANDROID

Autora: Rocío Ortega Olivas

Tutor: Dr. Alberto Brunete González

Director: Dr. Luis Pedraza Gómara

Marzo 2013

”*People are not remembered by the number of times they fail but for the number of times they succeed”*

”Las personas no son recordadas por el número de veces que fracasan sino por el número de veces que tienen éxito”

Thomas Alba Edison, 1927.

Título: Tratamiento de imágenes, extracción de características y "*matching*" en plataformas android

Autora: Rocío Ortega Olivas

Director: D. Alberto Brunete González
Codirector: Dr. Luis Pedraza Gómara

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 19 de marzo de 2013 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL:

SECRETARIO: PRESIDENTE:

Agradecimientos

A mi mis tutores, D. Luis Pedraza Gómara y D. Alberto Brunete González, por su paciencia y tiempo dedicado. Gracias por estar siempre disponibles para echarme una mano y por brindarme la oportunidad de realizar este proyecto, sin vuestro valioso tiempo y conocimientos no hubiese sido posible llevarlo a cabo.

A mi familia que me lo han dado todo y no han dejado de creer ni un segundo en mí.

A mis amigos, que siempre han estado ahí.

A todos los profesores que de alguna manera han contribuido a mi formación, y de manera especial, aquellos que me han hecho disfrutar de lo aprendido.

Índice general

Índice figuras

Índice de tablas

Resumen

En este proyecto se estudiará cómo de eficientes son los dispositivos móviles Android para el procesamiento de imágenes, así como una nueva técnica para llevar a cabo el *matching* entre imágenes (utilizando Teoría de Grafos).

Se tratarán conceptos básicos de tratamiento de imágenes, repasando qué es un descriptor de imagen, cuáles son los más utilizados y cómo trabajar con ellos en proyectos de desarrollo, junto con otras funciones típicas del tratamiento de imágenes.

Por otro lado, se estudiará cómo funciona la plataforma sobre la que trabajaremos (Android), cuál es su arquitectura y cómo se puede utilizar la librería por excelencia para trabajar con imágenes en el campo del desarrollo de aplicaciones (OpenCV). Todo ello, para terminar estudiando cómo tratar la información que obtenemos de una imagen y relacionarla mediante Teoría de Grafos para realizar el *matching*.

Para poder aplicar todos los temas tratados se creará una aplicación Android, importando la librería OpenCV en la que se podrá comprobar cómo de eficientes son estos terminales y cómo es posible aplicar Teoría de Grafos a la información que nos aporta una imagen para relacionarla con otra.

Abstract

In this project, the efficiency of Android mobile devices for image processing will be analyzed, as well as a new technique to perform matching between images (using Graph Theory)

Basic concepts of image processing will be studied, reviewing what is an image descriptor, which are the most used and how to work with them in development projects. Other typical image processing functions will be studied as well.

In addition, the platform used to develop the project (Android), its architecture and how we can use the OpenCV library for working with images in the field of application development will be covered. All this, to finish studying how to treat the information obtained from an image and link it using Graph Theory to perform the matching.

To implement all these goals an Android application will be created, importing the OpenCV library in order to check how efficient are these terminals, and how it is possible to apply Graph Theory to the information provided by an image to match it to another one.

Capítulo 1. Introducción y objetivos

En este capítulo se introducirá el contexto del Proyecto Fin de Carrera y la motivación existente para realizarlo. Se explicará de forma general los avances actuales y herramientas disponibles para abordar proyectos de tratamiento de imágenes y de Visión Artificial mediante el uso de dispositivos Android.

1.1 Introducción

Los avances tecnológicos de las últimas décadas han hecho posible tener los dispositivos y la gran cantidad de contenidos de hoy en día. Todos estos avances han venido de la mano de la popularización del uso de Internet y la aparición de la telefonía móvil. Los dos, son creaciones relativamente recientes, que a pesar de no tener más de cuarenta años evolucionan a un ritmo imparable.

El nacimiento de Internet se remonta a los años sesenta, en el seno de laboratorios y universidades que trabajaban en proyectos militares. En plena guerra fría, Estados Unidos crea una red exclusivamente militar, con el objetivo de poder acceder a información militar desde cualquier punto del país ante un hipotético ataque ruso. Por otro lado, la historia del teléfono móvil se remonta a los inicios de la Segunda Guerra Mundial, donde comienza a surgir la necesidad de comunicarse a distancia. El equipo Handie Talkie H12-16 de Motorola comienza a cubrir esta necesidad en un rango de frecuencias que no superara los 600kHz. Estas dos tecnologías de origen militar se han desarrollado fuertemente a lo largo de estos años cubriendo la mayor necesidad humana: la comunicación.

Hoy en día, Internet y telefonía móvil han encontrado su punto de convergencia. Una unión que nos abre un abanico de posibilidades en cualquier lugar y que supone un generador y receptor de una gran cantidad de contenidos.

Esta cantidad de información ha potenciado el incremento del número de aplicaciones que podemos utilizar tanto el ordenador como en los dispositivos móviles. Aplicaciones que tienen como requisito común, gestionar la información de manera eficiente.

Sea cual sea la información a tratar, todo contenido se representa de forma numérica, compuesto por características o descriptores de los que podemos extraer diferentes propiedades, permitiendo un tratamiento objetivo.

Si nos centramos en contenidos visuales, con el fin no sólo de almacenar la información, sino también de procesarla, conociendo sus características o descriptores estamos llevando a cabo técnicas de visión artificial, que como su propio nombre indica, tratan temas de visión de una manera "artificial", es decir, su objetivo es hacer que un ordenador "entienda" una escena o características de una imagen con el fin de realizar diversas tareas a través de su procesamiento, tales como el reconocimiento de objetos o mapeo de una escena para generar un modelo tridimensional.

Todas estas tareas han supuesto grandes retos en la computación, a pesar de que muchas de ellas pueden ser realizadas por los seres humanos de forma natural. Acciones tales como reconocer un objeto o detectar patrones de movimiento, han constituido grandes logros; logros que han tardado en llevarse a cabo debido a limitaciones tanto software como hardware. Ya que, aunque el inicio de las técnicas de visión artificial se remontan, desde el punto de vista práctico, al año 1961, cuando Larry Roberts desarrolló una aplicación para analizar una estructura de bloques a través de la información recogida por una cámara, no es hasta comienzos de los años 90 cuando comienzan a aparecer ordenadores con la velocidad de cómputo suficiente para procesar imágenes de forma ágil. Y, aunque hoy en día, muchas de las soluciones implementadas mediante esta técnica no sean la mejor solución para determinados problemas, los avances de los últimos años han permitido que, sin darnos cuenta, estemos rodeados de numerosas aplicaciones de este tipo. Aplicaciones que abarcan campos muy diversos, desde medicina para el diagnóstico por imagen hasta aplicaciones para el control de calidad de procesos industriales.

Gracias a la capacidad que ofrece los dispositivos móviles actuales y las nuevas aplicaciones que tratan sobre temas de visión artificial surge este proyecto, que aquí comienza. Un proyecto, por tanto, basado en la técnicas de procesamiento de imagen llevadas a un entorno móvil.

1.2 Motivación del Proyecto Fin de Carrera

Los avances que han tenido lugar en el ámbito de la visión por ordenador han estado motivados por los avances en la capacidad de cómputo, capacidad que ha sido extrapolada a dispositivos con dimensiones cada vez más reducidas.

En la actualidad, existe un gran interés en torno a las capacidades de los dispositivos móviles y de cómo llevar a estos herramientas que utilizamos día a día. Prueba de ello son las numerosas aplicaciones que han aparecido, que ya no sólo se ocupan de temas relacionados con las funcionalidades básicas que ofrecían los terminales hace menos de siete años, o de las relacionadas con temas de oficina, con el fin poder leer documentos, consultar el correo electrónico o "realizar" el trabajo que se podría llevar a cabo en un ordenador; sino, que también encontramos aplicaciones relacionadas con temas de salud, de deportes, de entretenimiento o de localización.

La cuestión principal en la que se ahonda en este proyecto es cómo de eficaz es el procesamiento de imágenes en estos nuevos terminales: ¿Cómo de rápido podemos extraer las características de una imagen? ¿Qué limitaciones presentan? ¿Es posible reducir el tiempo de procesamiento en el emparejamiento de imágenes con técnicas basadas en Teoría de Grafos?

1.3 Objetivos del proyecto

Este proyecto tiene como objetivo principal estudiar el tiempo de procesamiento ante diferentes técnicas de tratamiento de imágenes y el estudio de un método diferente

para realizar la correspondencia entre imágenes, lo cual posee infinidad de aplicaciones, tales como la búsqueda de patrones dentro de una imagen o realizar el solape entre varias imágenes mediante los puntos correspondientes (panorámica).

El problema de la correspondencia entre imágenes se ha abordado tradicionalmente mediante dos técnicas principales:

1. Búsqueda de patrones, considerando como patrón un elemento característico en una de las imágenes. El método más sencillo para llevarlo a cabo es mediante comparación de plantillas (template matching), utilizado en aplicaciones de seguimiento, detección o reconocimiento de objetos. Pero con desventajas tales como su coste computacional, o(WHwh) y una gran sensibilidad a variaciones de escala, rotación o deformaciones 3D de los objetos.

2. Flujo óptico, técnica de análisis de imágenes que se aplica en secuencias de vídeo. Una forma sencilla de implementación es mediante la división de la imagen en bloques y búsqueda de correspondencia para cada bloque. Es una técnica muy lenta, inviable para realizar en tiempo real, pero permite comprender mejor la información contenida y es menos sensible a cambios en la escena.

En este proyecto se estudiará la posibilidad de realizar esta correspondencia mediante la técnica del máximo cliqué aplicada a grafos, mediante la previa extracción de las características de las dos imágenes.

Con el fin de cumplir los objetivos del proyecto, se implementará una aplicación para Android, la cual abarcará los siguientes puntos:

- Estudio del tiempo de procesamiento al aplicar distintas técnicas de procesamiento de imágenes propias de OpenCV sobre una imagen almacenada en el terminal.

- Estudio del tiempo de procesamiento para establecer la correspondencia entre dos imágenes utilizando las características SURF de ambas y relacionarlas mediante Teoría de Grafo (técnica de Máximo Cliqué).

Para lograr este objetivo final, se tienen que ir alcanzando antes una serie de pasos no menos importantes. Las fases que se desarrollarán a lo largo de este trabajo, y que conforman los objetivos del mismo, son las siguientes:

Realizar un estudio de la plataforma Android: estudio general del sistema operativo de Google.

Realizar un estudio de los métodos de extracción de información a través de los descriptores más comunes en el tratamiento de imágenes.

Estudio de la posibilidad de utilizar la librería OpenCV, librería en código C, en una aplicación Android.

Estudio de los conceptos básicos de Teoría de Grafos y técnicas de Máximo Cliqué.

Estudio de la posibilidad de aplicar Teoría de Grafos para el matching entre dos imágenes a través de las características extraídas utilizando el descriptor seleccionado.

Análisis y desarrollo de la aplicación en Android: estudio del diseño a implementar. Estudio de las funciones del programa, sus casos de uso y la interfaz gráfica. Desarrollo de la aplicación sobre la plataforma Android.

Estudio de la capacidad de respuesta del terminal y de los resultados obtenidos en función de las imágenes procesadas.

1.4 Fases de desarrollo

De forma general, se ha dividido la metodología de trabajo en cinco bloques:

- Familiarización. En esta primera etapa del proyecto se ha llevado a cabo una primera toma de contacto con los temas relacionados con visión artificial.

- Investigación. Una vez que se tienen los conceptos básicos sobre visión artificial, se ha llevado a cabo un estudio sobre el estado del arte de las aplicaciones móviles

desarrolladas. Al mismo tiempo, se ha investigado sobre las herramientas que se utilizan para realizar el procesamiento de imágenes y cómo se llevan a cabo las tareas de extracción de características y emparejamiento (*matching*).

- Implementación y desarrollo. La tarea principal no es sólo investigar, sino evaluar las capacidades que presentan los dispositivos móviles en este campo, y para ello, como ya se mencionó anteriormente, se desarrollará una aplicación Android.

- Estudio de resultados y formalización de las conclusiones. Finalmente se realizará un estudio exhaustivo de los resultados obtenidos y de las conclusiones.

- Escritura del proyecto. Si bien esta tarea se ha desarrollado de forma continua a lo largo del proyecto, gran parte de este documento se ha completado al finalizar la etapa de desarrollo.

1.5. Medios empleados

1.5.1. Recursos Hardware

Para realizar este proyecto se han empleado una serie de recursos. La ejecución de la aplicación durante las pruebas se ha realizado en el simulador Dalvik y en un dispositivo con sistema operativo Android 2.1 de gama media.

La máquina virtual Dalvik

En Android, todas las aplicaciones se programan en el lenguaje Java y se ejecutan mediante un simulador o máquina virtual denominada Dalvik, diseñada y adaptada a las peculiaridades propias de los dispositivos Android y cuyas características podrán ser modificadas.

En nuestro caso, la configuración de la máquina virtual es la siguiente:

- Memoria RAM de 524.3 MB.
- Android 2.1 API Level-7

Ejecutado en un ordenador Intel Core 2 Duo con una velocidad de procesador de 2GHz y 2 GB de RAM.

Dispositivo utilizado

HTC Wildfire

Para estudiar el tiempo de procesamiento para el tratamiento de imágenes utilizaremos un dispositivo de gama media con las siguientes características [28]:

- Pantalla táctil capacitiva, con una resolución de 240x320 QVGA.
- Velocidad de procesamiento de CPU de 528 MHz.
- Versión Android 2.1 (Éclair) con HTC Sense.
- Almacenamiento ROM de 512 MB.
- Almacenamiento RAM de 384 MB.
- Cámara de 5 Megapíxeles.

Fig. 1. HTC Wildfire

Aunque se utilizó la librería propia de OpenCV durante el estudio de las posibilidades que ofrecía, finalmente se optó por utilizar una librería que porta OpenCV para Android y que no incluye todas las funciones de la última versión, presentando un tamaño menor, de tal forma que pueda ser utilizada en dispositivos con menor memoria RAM y menor velocidad de procesador.

Ordenador portátil

Utilizado para el desarrollo de la aplicación a través del entorno de desarrollo Eclipse, con las siguientes características:

- Sistema Operativo principal: Mac Os 10.6.7

- Sistema Operativo secundario (Máquina Virtual): Windows XP

- Procesador Intel Core 2 Duo de 2GHz.

- Memoria RAM: 2 Gb 1067MhHz DDR3

1.5.2. Recursos software

A parte de los recursos hardware se han utilizado una serie de herramientas software y lenguajes de programación.

Entorno de desarrollo Eclipse.

Eclipse[27], entorno de desarrollo integrado (IDE) de código abierto multiplataforma basado en JAVA. Fue desarrollado originariamente por IBM como sucesor de su familia de herramientas VisualAge y actualmente desarrollado por la Fundación Eclipse, organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto. Posee una arquitectura extensible mediante *plugins*, entre los que encontramos el correspondiente para Android[18, 19].

Es el IDE oficial del proyecto Android y soporta la edición de ficheros JAVA, XML e incluso de recursos gráficos, facilitando así la programación en esta plataforma.

1.5.3. Lenguajes utilizados

C ++ (OpenCV)

Lenguaje de programación que surge ante la necesidad de extender el exitoso lenguaje de programación C, introduciendo la programación basada en objetos. Desde el punto de vista de lenguajes orientado a objetos es un lenguaje híbrido, al que además se le añadieron facilidades de programación genérica, que sumada a los paradigmas ya presentes (programación estructurada y orientada a objetos) lo convierten en un lenguaje multiparadigma.

JAVA

Lenguaje de programación oficial para la plataforma Android, orientado a objetos, desarrollado por SUN Microsystems a principios de los 90, a diferencia de C/C++ es un modelo de objetos más simple que elimina herramientas de bajo nivel, como la gestión de memoria. Sin embargo, lo más característico de JAVA es que es un lenguaje multiplaforma.

XML

XML, siglas en inglés de eXtensible Markup Language (‘lenguaje de marcas extensible’), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML (Standard Generalized Markup Language) que permite definir la gramática de lenguajes específicos[20]. En definitiva, es el estándar utilizado para el intercambio de información y que es utilizado en Android para definir la apariencia.

1.5.4 Librerías utilizadas

OpenCV

OpenCV (Open Source Computer Vision Library), conjunto de bibliotecas en C y C++ de código libre ("Open Source BSD") orientadas a visión por computador. Posee más de 500 funciones, que incluye funciones en lenguaje nativo para aplicaciones de visión artificial. Esta librería aparece descrita en el Anexo, así como las herramientas necesarias para utilizarla.

GraphT

Librería JAVA para la construcción de grafos y empleo de distintos tipos de algoritmos. Permite construir grafos cuyos vértices pueden ser de distinto tipo. Toda la información de las funciones a emplear, así como el descargable de la librería para incorporar a Eclipse se encuentra en su web [22].

1.6 Estructura de la memoria

Este documento pretende recopilar toda la información relativa al desarrollo del proyecto, por tanto, aquí se encuentran todos los detalles técnicos y funcionales del mismo, que se han estructurado para atender a dos tipos de lectores: (1) personal técnico que se enfrente a un posible versión futura del proyecto o que trabaje en temas relacionados, para facilitar la transmisión de todo el conocimiento del diseño y del desarrollo adquirido; (2) profesionales no necesariamente expertos en la materia que

quieran tener una visión global del proyecto para comprender y valorar el trabajo realizado.

De esta forma general, la estructura de la memoria es la siguiente:

- Introducción y objetivos. Se ofrece una introducción al proyecto, presentándose las motivaciones y los objetivos que se pretenden conseguir.
- Estado del arte. Se realiza un estudio de la situación actual de los proyectos y aplicaciones Android bajo la perspectiva de este proyecto. A partir de este análisis, se sacan las conclusiones que sirven como base para la implementación.
- Diseño y desarrollo del sistema. Recoge el conjunto de requisitos que ha de cumplir el sistema a implementar, la arquitectura diseñada para cubrir estos requisitos y la implementación realizada.
- Gestión del proyecto. Detalla todos los aspectos relacionados con la gestión del proyecto, tales como los recursos necesarios, plan de trabajo y una estimación de los costes de implementación.
- Conclusiones y trabajo futuro. Recoge los resultados logrados y las conclusiones obtenidas tras la realización del proyecto, analizando el nivel de consecución de los objetivos iniciales y presentando las líneas futuras de ampliación y mejora.

Capítulo 2. Estado del Arte

Antes de profundizar en el tema que aborda el proyecto es necesario entender la plataforma en la que se desarrolla y qué influencia tiene esta tecnología en el mercado. Todo ello, para finalmente conocer qué proyectos están relacionados con el mismo y qué herramientas nos pueden servir de ayuda.

2.1 Introducción

La principal pregunta a la que tenemos que responder para entender porqué se ha elegido la plataforma para el proyecto es ¿Qué es Android? y sobre todo, ¿qué influencia tiene hoy en día en el mercado?

Android no es más que la forma de afrontar la telefonía por parte de Google, que entiende estos dispositivos como una forma de estar constantemente conectado a Internet y a sus herramientas. Pero esta visión de Google ha ido más allá, ya que el hecho de desarrollar un Sistema Operativo libre ha permitido una importante penetración en el mercado de dispositivos inteligentes a costes relativamente bajos, los cuales tienen a su disposición una gran cantidad de aplicaciones que extienden casi sin límites la experiencia del usuario.

Aplicaciones que han ido aumentando a medida que los dispositivos con esta plataforma se extendían en el mercado, lo cual no ha sido difícil gracias a las principales característica de este Sistema Operativo, libre, gratuito y multiplataforma.

Según la compañía de investigación "Nielsen" la cuota de mercado de dispositivos Android ha pasado de un 2.9% en 2009 a superar el 51% en julio de 2012, seguido del 34% correspondiente a Apple (ver figura 2).

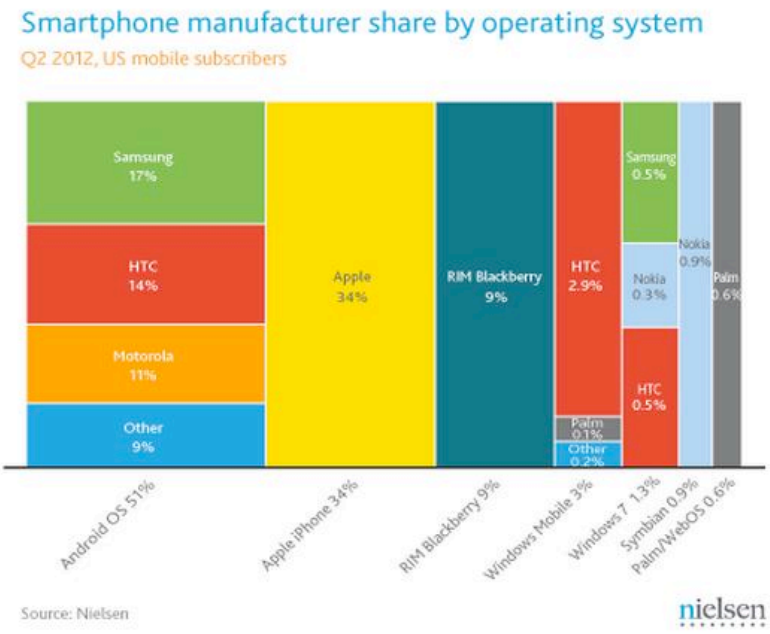

Fig. 2. Cuota de mercado de dispositivos (nielsen)

Esta evolución imparable ha estado motivada por el creciente número de aplicaciones, ya que esta plataforma brinda la posibilidad de un fácil desarrollo y distribución, no como sucede con su mayor competidor, Apple. En la siguiente figura se puede ver el número de aplicaciones para la plataforma Android, la cual es comparable con la de su otro gran competidor.

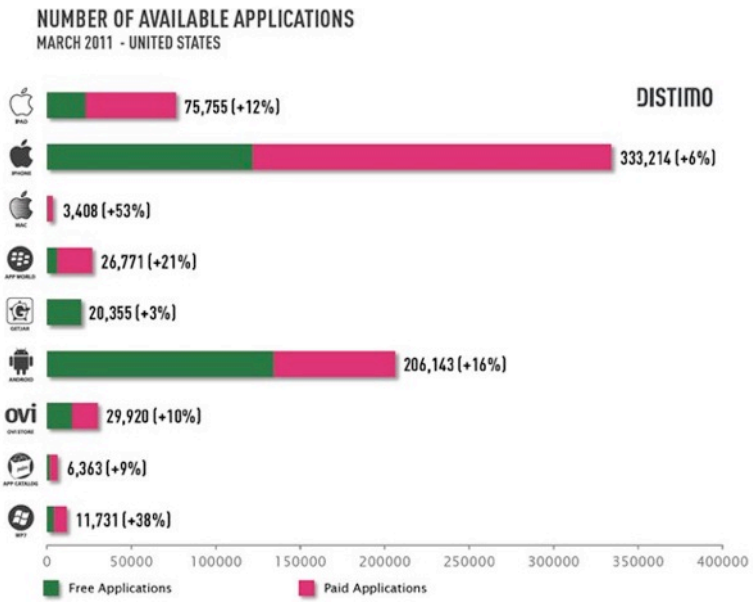

Fig. 3. Comparativas de aplicaciones disponibles en las distintas plataformas.

Vista la influencia que tiene en el mercado, la necesidad de realizar el proyecto sobre esta tecnología puede entenderse como la necesidad de estudiar si estos dispositivos son capaces de permitir implementar funciones de visión artificial, para extender aún más las posibilidades ofrecidas al usuario.

2.2 Aplicaciones de tratamiento de imágenes en terminales móviles

Cabe destacar que en los últimos años, se han lanzado al mercado distintos proyectos de tratamiento de imágenes en terminales móviles; quizás los más sorprendentes son todos aquellos relacionados con Realidad Aumentada, en los que se puede ver en tiempo real sobre las imágenes que captura la cámara la información de interés. Un ejemplo de ello, es la aplicación Word Lens[10] para iPhone, traductor en tiempo real de las imágenes capturadas por la cámara (ver figura 4).

Fig. 4. Traductor Word Lens

O, la aplicación Layar [11] (figura 5), aplicación que ya se puede encontrar para diversas plataformas, Android, iPhone u OVI NOKIA, mediante la cámara y el GPS añade una capa de información por encima de la real para la localización de puntos de interés. La aplicación permite escanear todo lo que está a nuestro alrededor: menús restaurante, tienda de comestibles, revistas, carteles, vallas publicitarias, hoteles, establecimientos comerciales, y mucho más.

Fig. 5. Layar

Otra de las aplicaciones más importantes es iOnRoad Augmented Driving Lite (figura 6). La mayoría de los teléfonos Android se pueden utilizar como navegadores mientras se conduce. La aplicación iOnRoad Augmented Driving Lite, además, ayudada a ir con cuidado en la carretera controlando y vigilando la distancia de seguridad entre tu coche los demás.

Fig. 6. iOnRoad Augmented Driving

Similar a las anteriores, pero con finalidad diferente, encontramos Wikitude World Browser (figura 7), una enciclopedia online que comparte información en realidad aumentada. Esta aplicación permite ver información adicional acerca de los lugares con sólo mirarlos a través de la pantalla del smartphone.

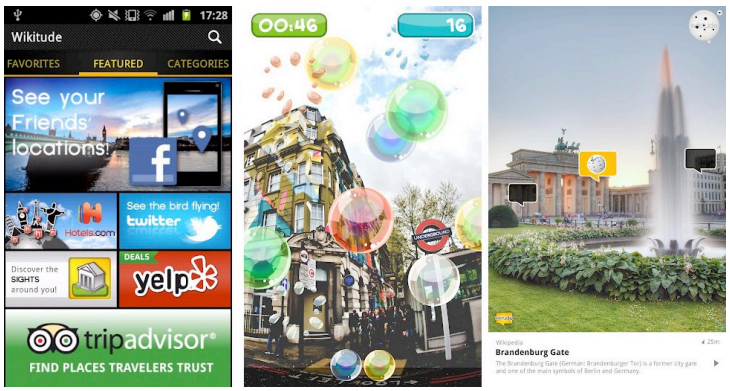

Fig. 7. Wikitude

2.3 Aplicaciones de tratamiento de imágenes en terminales móviles basadas en OpenCV

OpenCV (Open Source Computer Vision Library) es un conjunto de bibliotecas en C y C++ de código libre ("Open Source BSD") orientadas a visión por computador. Posee más de 500 funciones, que incluye funciones en lenguaje nativo para aplicaciones de visión artificial.

Entre las aplicaciones que podemos encontrar desarrolladas con OpenCV destacamos las siguientes:

- Detector de coches [8]

Fig. 8. Detección de coches

- Reconocimiento facial[9]

Fig. 9. Detector facial en Android con OpenCV

A pesar de las distintas aplicaciones que se pueden encontrar, cabe destacar que es un ámbito que todavía no está lo suficientemente explotado, ya que la librería OpenCV, librería gratuita para el tratamiento de imágenes, contiene más de 500 funciones; no comparable con el número de aplicaciones desarrolladas y con lo que todavía se puede obtener de esta extensa librería.

Al margen de todos los proyectos relacionados con el tratamiento de imágenes para terminales móviles, se destacarán aquellos que sirvan para estudiar aspectos de este proyecto. Se destacan los siguientes:

- CVCamera[1]. Aplicación que procesa los frames extraídos de la cámara y muestra en tiempo real la extracción de características SURF, FAST y STAR. También lleva a cabo el proceso de calibración. Este proyecto se encuentra dentro las aplicaciones de ejemplo de la librería de OpenCV para Android[2]. Gracias a este proyecto se ha podido analizar la capacidad de los descriptores SURF para extraer información.

Fig. 10. Extracción de características FAST con CVCamera

- 360pano[3]. Aplicación basada en la anterior, con nuevas funcionalidades, entre ellas, obtener una imagen panorámica con la información de las características SURF, FAST o STAR ya extraídas, aunque posee limitaciones a la hora de capturar las imágenes. Para realizar la correspondencia de puntos emplea la técnica del flujo óptico.

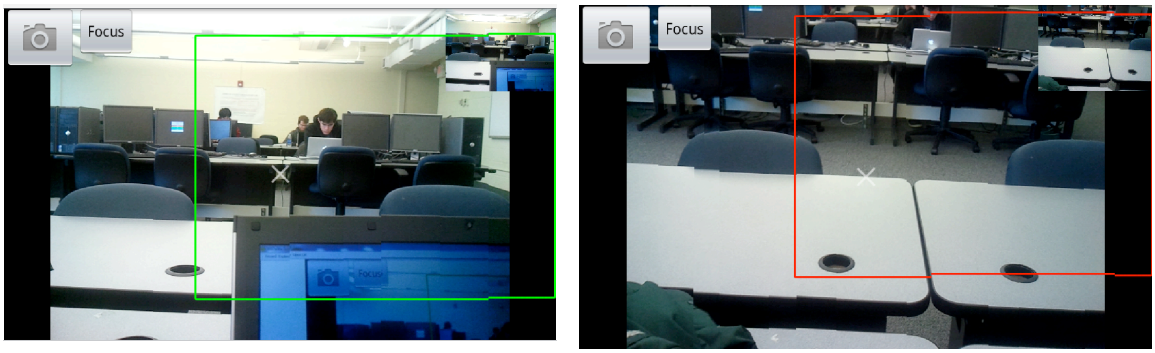

Fig. 11. Captura de imágenes con 360 pano

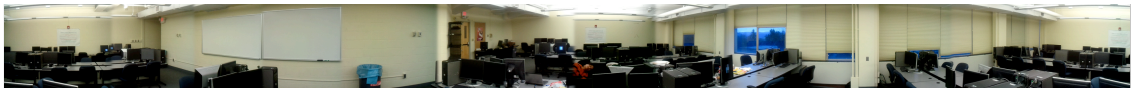

Fig. 12. Panorámica con 360 pano

-RobotView[4]. Aplicación de la misma compañía que CVCamera. Elabora una imagen panorámica a partir de las imágenes capturadas con la cámara. En esta aplicación no existe restricción a la hora de tomar la fotografía, es decir, no tienen que estar necesariamente alineadas. No se dispone del código fuente de esta aplicación, ni detalles de cómo realiza tal proceso, por lo que no se puede analizar con mayor detalle.

Fig. 13. Panorámica realizada con RobotView

A pesar de que las tres aplicaciones anteriores son las más destacadas en relación con el tema tratado, existen otras aplicaciones basadas en el procesamiento de imágenes con OpenCV.

Capítulo 3. Marco teórico

En este capítulo se explicará el concepto de descriptor de imagen y cuáles son los dos más utilizados: SIFT y SURF.

3.1. Descriptores de imagen

Cada imagen se puede descomponer en unidades mínimas de información, llamadas píxeles. Los píxeles constituyen la equivalencia descriptiva entre la información que almacenamos para la imagen y las características que son apreciables para el ser humano.

Para satisfacer la necesidad de describir los contenidos visuales surgen los "descriptores visuales", que como su propio nombre indica, describen las características visuales de los contenidos dispuestos en imágenes o vídeos.

Estos descriptores visuales se clasifican en dos tipos:

- Descriptores de información general, contienen descriptores de bajo nivel, proporcionando una descripción a cerca del color, formas, regiones, texturas y movimiento.
- Descriptores de información de dominio específico, proporcionan información acerca de los objetos y eventos que van apareciendo en la escena, como por ejemplo el reconocimiento facial.

Estos dos tipos de descriptores hacen referencia a los distintos niveles de profundidad en la descripción del contenido. Siendo los de información general aquellos que se encuentran en el nivel más bajo de abstracción, haciendo referencia a las

características básicas de una imagen, tales como texturas, colores o formas. En un nivel superior encontramos los descriptores de dominio específico, los cuales llevan a cabo una descripción semántica de las imágenes.

Dentro de los descriptores de información general, podemos clasificarlos según el nivel de aplicación dentro de la imagen:

Descriptores globales. Resumen el contenido de la imagen en un único vector o matriz de características. Esto posee como ventaja principal, la encapsulación de una gran cantidad de información en una única estructura, lo que suponen un menor coste computacional. Un ejemplo sería el histograma de color.

- Descriptores locales. Proporcionan una descripción local de la imagen, centrándose en regiones, que previamente pueden ser calculadas o identificadas, constituyendo un vector de características que tiene en cuenta los puntos de esa región y los adyacentes. Normalmente las regiones descritas reciben el nombre de puntos de interés o *keypoints*. El descriptor estará por tanto, formado por todos los vectores de puntos de interés calculados en una imagen. Un ejemplo de este tipo de descriptor sería SIFT.

Una vez introducido el concepto de descriptor, se explicarán los descriptores que se ocupan de extraer las características de una imagen (descriptores de imagen), y más concretamente el tipo que se estudiará en este proyecto, los descriptores locales.

Idealmente un buen descriptor posee una serie de propiedades [23], tales como repetibilidad, eficiencia o exactitud.

Aquellos puntos que vayan a ser procesados, deben ser lo más estables posibles y permanecer invariantes ante posibles cambios de rotación, orientación o escala, de tal forma que puedan ser encontrados en un gran número de imágenes consecutivas.

A continuación se estudiarán dos de los descriptores más empleados, SIFT y SURF.

3.1.2 Detector SIFT (Scale Invariant Transform)

Fue desarrollado por David G. Lowe, en 1999, como un algoritmo capaz de detectar puntos de interés estables de una imagen presentando un buen rendimiento en relación velocidades de cálculo y precisión.

Los puntos extraídos son invariantes frente a diferentes transformaciones de traslación, escala, rotación, iluminación y transformaciones afines.

El algoritmo de SIFT se compone principalmente de cuatro etapas:

1. **Detección de Extremos en el Espacio Escala.** En esta primera etapa se realiza una búsqueda sobre las diferentes escalas y dimensiones de la imagen identificando posibles puntos de interés, invariantes frente a cambios de orientación y escalado. Todo esto se lleva a cabo mediante la función DoG (Difference-of-Gaussian).

Cada imagen será filtrada mediante una Gaussiana, de modo que los puntos de interés SIFT corresponden a los máximos y mínimos locales resultante de obtener la diferencia entre varios filtros Gaussianos a escalas diferentes. De modo que el espacio escala de una imagen se define como la función L(x,y,σ), resultante de la convolución de la imagen de entrada con una Gaussina de escala variable:

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

Donde tenemos que I es la imagen de entrada y G la Gaussiana de escala variable, la cual es igual a:

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{\sigma^2}}$$

Para conseguir una detección eficiente de los puntos estables, este detector sugiere utilizar la convolución de una diferencia de Gaussianas con la imagen:

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma)$$

Donde D(x,y,σ), es la diferencia de los resultados de difuminar la imagen de entrada con una Gaussiana de escala kσ y σ.

Los motivos que llevan a la utilización de esta función D, también denominada DoG, son mayor eficiencia al tratarse simplemente de la resta de dos imágenes y una aproximación más precisa del Laplaciano normalizado de una Gaussiana.

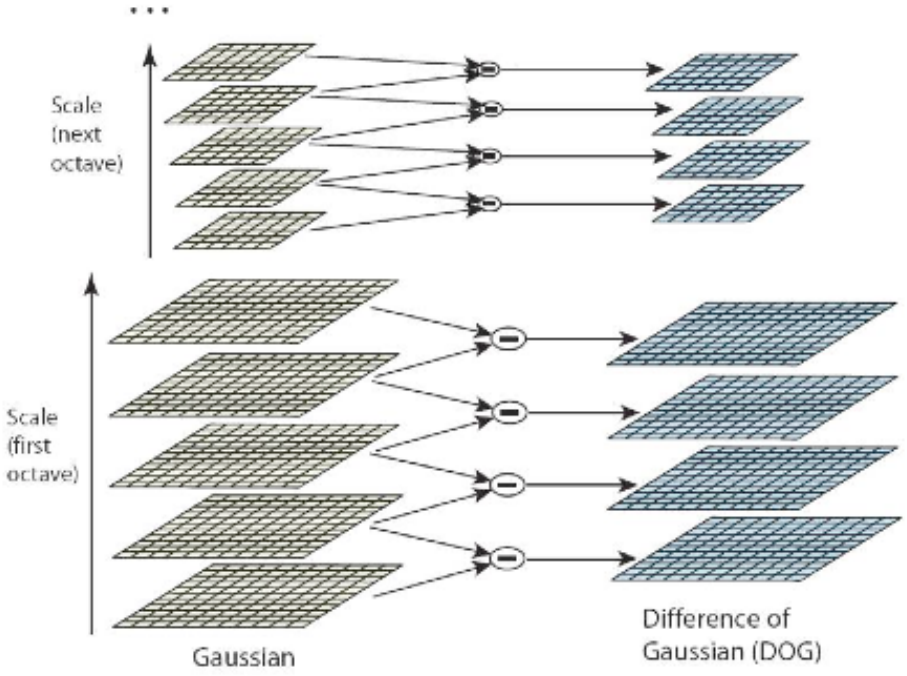

Fig. 14. Creación del espacio-escala Gaussiano.

En la figura 14 se muestran las imágenes difuminadas a diferentes escalas, y las imágenes resultantes de aplicar la diferencia de Gaussianas.

Las imágenes resultantes de la convolución son agrupadas por octavas, la cual corresponde a doblar el valor de σ. Cada octava es dividida en un número s de intervalos, y el valor del factor k, el cual determina la separación de las imágenes espacio-escala y está definido como $k = 2^{\frac{1}{s}}$, es seleccionado de forma que obtengamos un número fijo de imágenes borrosas por octava.

Finalmente, los puntos de interés son identificados como los máximos o mínimos locales de las imágenes DoG a través de diferentes escalas. Cada píxel en las imágenes DoG es comparado con sus 8 vecinos en su misma escala, más los correspondientes 9 vecinos en escalas vecinas (superior e inferior). Obteniendo con ello, un primer grupo de posibles candidatos de puntos de interés.

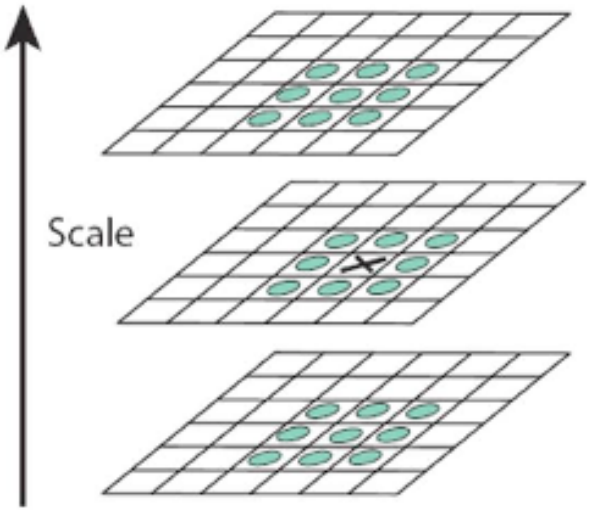

Fig. 15. Localización de máximos y mínimos locales. Comparación con varias escalas.

2. **Localización de los Puntos de Interés**. Para ello, se aplica una medida de estabilidad sobre todos ellos, con el fin de descartar aquellos que no sean adecuados.

Una vez que los puntos de interés han sido calculados, se realiza un estudio de su estabilidad. Aquellos que no están sobre los bordes o aquellos con bajo contraste son vulnerables al ruido y por lo tanto, no podrán ser detectados ante pequeños cambios de iluminación o variación del punto de vista de la imagen. Para descartarlos, sobre cada punto de interés haremos lo siguiente:

- Mediante la interpolación de los datos cercanos se determina de manera precisa su posición.

- Aquellos con bajo contraste son eliminados. Para ello se aplica un proceso de umbralización, por el cual aquellos puntos cuyo valor sea menor que el fijado como umbral serán eliminados.

- Aquellos que se correspondan a bordes también deben eliminarse, ya que conllevan un alto grado de inestabilidad incluso ante pequeños ruidos. Para llevar a cabo su eliminación, se utiliza la propiedad de la función DoG atendiendo a la gran curvatura que presenta en la dirección paralela al borde y la pequeña curvatura que se observa en la dirección perpendicular. Estas respuestas tan característica se pueden estudiar mediante el cálculo de la matriz Hessiana sobre la localización y escala del punto de estudio:

$$H = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}$$

Donde D es la imagen DoG (x,y,σ) respecto de la escala s. Las derivadas se calculan mediante la resta del valor de los puntos vecinos. Se puede demostrar que la siguiente desigualdad permite la localización de los puntos en los bordes:

$$\frac{\left(\frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2}\right)^2}{\left(\frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2}\right) - \left(\frac{\partial^2 D}{\partial x \partial y}\right)^2} < \frac{(r+1)^2}{r}$$

Por tanto aquellos puntos que no satisfagan dicha desigualdad serán descartados debido a su inestabilidad.

3. **Asignación de la Orientación**. Se asignan una o más orientaciones a cada punto de interés extraído de la imagen, basándose en las direcciones locales presentes en la imagen gradiente, lo que nos asegura la invarianza sobre estas transformaciones.

Para determinar la orientación del punto de interés, se calcula un histograma del gradiente de la orientación en los puntos vecinos al punto de interés, utilizando para ello la imagen Gaussiana más cercana al punto de interés.

Para cada imagen muestreada la magnitud del gradiente (m(x,y)) y la orientación (θ(x,y)) son las siguientes:

$$m(x,y) = \sqrt{L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \arctan g \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}$$

A la contribución de cada píxel vecino se le asigna un peso en función de la magnitud del gradiente y a una Gaussiana con σ 1.5 veces la escala del punto de interés. Los picos del histograma corresponderán a las orientaciones dominantes. Se crea aparte un punto de interés con la dirección relativa al máximo del histograma y con cualquier dirección que tenga al menos el 80% del valor máximo.

4. **Descriptor del Punto de Interés**. Esta última etapa hace referencia a la representación de los puntos de Interés como una medida de los gradientes locales de la imagen en las proximidades de dichos puntos clave y respecto a una determinada escala. Cada punto de interés corresponde a un vector de características compuesto de 128 elementos, que le confieren una invarianza parcial a deformaciones de forma así como a cambios de iluminación.

En las etapas anteriores hemos asignado a cada punto de interés una posición, una escala y una orientación. Por tanto, en esta etapa lo que haremos es crear descriptores usando los histogramas de orientaciones. Los descriptores creados deberán ser suficientemente distintivos y permanecer invariantes frente a posibles cambios.

El descriptor se computa como un conjunto de histogramas de orientación creados sobre una región de muestras 4x4 en los píxeles vecinos. Los histogramas de orientación se construyen en función de la orientación del punto de interés, proviniendo los datos de orientación de la imagen Gaussiana más cercana en escala a la del punto de interés. Al igual que antes, la contribución de cada píxel es ponderada por la magnitud del gradiente y por una Gaussiana de escala 1.5 veces la del punto de interés.

Cada histograma contiene ocho referencias, y cada descriptor contiene un conjunto de cuatro histogramas alrededor del punto característico. Esto nos lleva a que el

descriptor SIFT está formado por un vector de tamaño 4x4x8=128 elementos. La normalización de este vector le dota de invarianza frente a cambios de iluminación.

Resumiendo, cada vector SIFT contendrá la siguiente información:

- Posición (x,y) en la imagen.
- Orientación.
- Escala.
- Descripción de su entorno por medio de un conjunto de gradientes.

3.1.3. Speeded Up Robust Features (SURF)

El descriptor SURF, cuyo acrónico hace referencia al título, Speeded-Up Robust Features, fue desarrollado por Hebert Bay et. Al. [23] como un detector de puntos de interés y descriptor robusto. El descriptor SURF guarda cierta similitud como la filosofía del descriptor SIFT. Los autores, sin embargo, afirman que presenta dos notables mejoras:

- La velocidad de cálculo es considerablemente superior sin ocasionar pérdida de rendimiento.

- Mayor robustez ante posibles transformaciones de la imagen.

Estas mejoras se consiguen mediante la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos, mientras continúan siendo suficientemente característicos e igualmente repetitivos.

Estas mejoras no son las únicas diferencias con respecto a SIFT, sino que además:

- La normalización o longitud de los vectores de características de los puntos de interés es considerablemente menor, concretamente de dimensionalidad 64 frente a los 128 de SIFT.

- El descriptor SURF siempre trata con la misma imagen, la original.

- Utiliza el determinante de la matriz Hessiana para calcular tanto la posición como la escala de los puntos de interés.

A continuación, vamos a describir los pasos necesarios para obtener los descriptores SURF.

Detección de puntos de interés

El detector SURF se basa en la utilización de la matriz Hessiana, mediante la cual consigue un buen rendimiento en cuanto a velocidad de cálculo y precisión. Concretamente utiliza el determinante de la matriz, para la localización y la escala de los puntos. Sin embargo, lo realmente novedoso de este descriptor es que en vez de usar una medida diferente tanto para elegir la posición como la escala, lo que hace es emplear el determinante de la matriz Hessiana en ambos casos. Por lo tanto, dado un punto, $p = (x, y)$ de la imagen *I*, la matriz Hessiana $H = (p, \sigma)$ en el punto p a la escala σ se define como:

$$H(p,\sigma) = \begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{yx}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix}$$

Donde $L_{xx}(p,\sigma)$ es la convolución de la derivada parcial de segundo orden de la Gaussiana $\frac{\partial^2}{\partial x^2} g(\sigma)$ con la imagen *I* en el punto *p*. Lo mismo ocurre para $L_{xy}(p,\sigma)$ y $L_{yy}(p,\sigma)$.

Los filtros Gaussianos son óptimos para el análisis del espacio-escala, debido a una serie de limitaciones, tales como la necesidad de ser discretizados, la no prevención del efecto aliasing, etc.; debido a esto, se ha implementado un nuevo tiempo de filtro, los filtro tipo caja (de su nombre en inglés box-filters). Estos aproximan las derivadas parciales de segundo orden de las Gaussianas y pueden ser evaluados de forma rápida

mediante imágenes integrales, independientemente del tamaño de éstas. Toda la información sobre imágenes integrales, la podemos encontrar detallada en [25,26].

Las aproximaciones de las derivadas parciales se denotan como D_{xx}, D_{xy} y D_{yy}. El determinante queda definido:

$$\det(H) = D_{xx}D_{yy} - (0{,}9D_{xy})^2$$

Donde el valor 0,9 está relacionado con la aproximación del filtro Gaussiano.

En la figura 16 se puede observar la representación de la derivada parcial de segurndo orden de un filtro gaussiano discretizado y la aproximación de la derivada implementada en SURF.

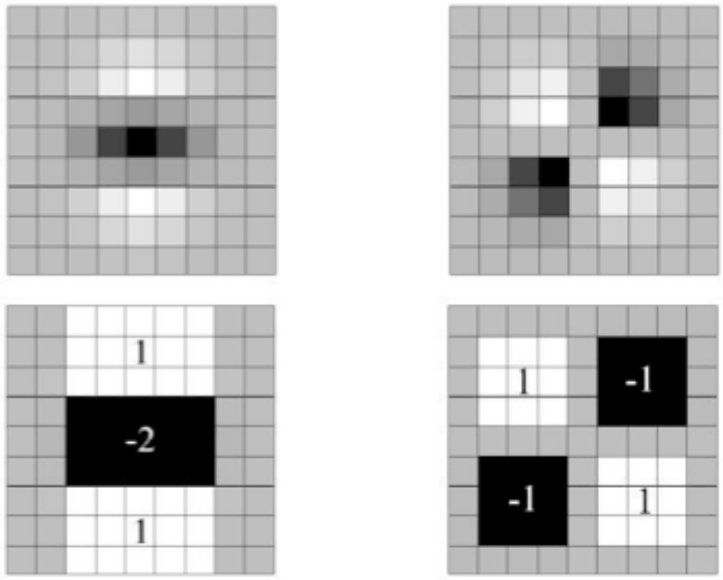

Fig. 16. Derivadas parciales de segundo orden de un filtro gaussiano y su aproximación.

La derivada parcial de segundo orden de un filtro gaussiano en la dirección xy se corresponde con las imágenes de arriba y sus respectivas aproximaciones las de abajo.

Los espacios de escala son a menudo implementados como pirámides de imágenes, en las que éstas son suavizadas repetidamente con un filtro Gaussiano y posteriormente submuestreadas para alcanzar un nivel más alto en la pirámide. En el detector SURF debido al uso de filtros de caja e imágenes integrales no se tiene que aplicar iterativamente el mismo filtro a la salida de una capa filtrada previamente, sino que se pueden aplicar dichos filtros de cualquier tamaño a la misma velocidad directamente en

la imagen original. De modo que el espacio escala es analizado por medio de ir elevando el tamaño del filtro, en vez de ir reduciendo el tamaño de la imagen. Las sucesivas capas se van obteniendo como resultado de aplicar filtros cada vez mayores.

El espacio de escala para el descriptor SURF, está dividido en octavas, donde cada octava está compuesta por un número fijo de imágenes como resultado de la convolución de la misma imagen original con una serie de filtros cada vez más grandes, como ya se mencionó anteriormente. En cada octava el incremento de tamaño del filtro es el resultado de doblar el incremento realizado en la octava anterior.

$$\text{Octava inicial: } 9x9 \xrightarrow{6} 15x15 \xrightarrow{6} 21x21 \xrightarrow{6} 27x27$$

$$\text{Siguiente octava: } 15x15 \xrightarrow{6} 27x27 \xrightarrow{6} 39x39 \xrightarrow{6} 51x51$$

$$\text{Siguiente octava: } 27x27 \xrightarrow{6} 51x51 \xrightarrow{6} 75x75 \xrightarrow{6} 99x99$$

Y así sucesivamente:

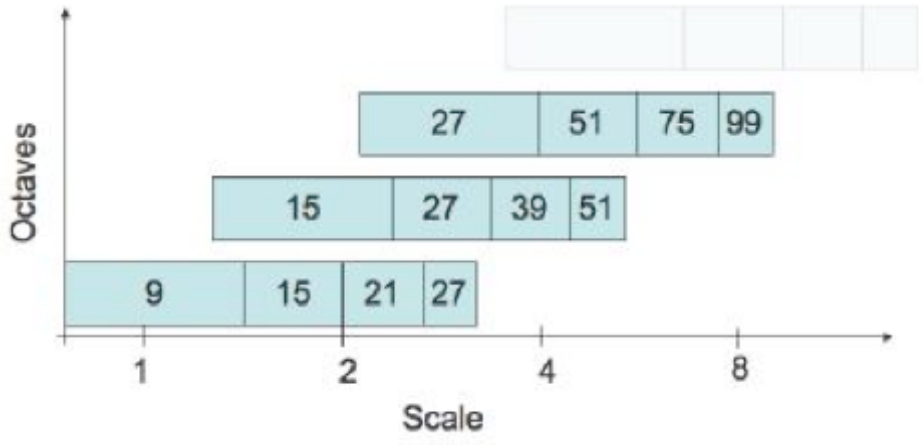

Fig. 17. Representación de la longitud de los filtros de diferentes octavas

Finalmente para detectar la localización de los puntos de interés en todas las escalas se procede mediante la eliminación de los puntos que no cumplan la condición de máximo en un vecindario 3x3x3. Entonces el máximo determinante de la matriz

Hessiana es interpolado en escala y el espacio de la imagen. Con esto, habríamos detectado los puntos de interés.

Asignación de la orientación

Una vez encontrados los puntos de interés, el siguiente paso es la asignación de la orientación al descriptor de cada punto (al asignar la orientación a los distintos puntos, se consigue invarianza ante la rotación). Para ello, se calculará la respuesta Haar en la dirección x e y:

Fig. 18. Filtros Haar empleados en SURF

En la figura 18, la imagen de la izquierda se corresponde con el filtrado Haar para el cálculo de la respuesta en el eje x y la de la derecha en el eje y. El color negro identifica el valor -1 y el blanco +1-

El área de interés para el cálculo es el área circular centrada en el punto de interés y de radio 6s, siendo s la escala en la que el punto de interés ha sido detectado.

La etapa de muestreo también depende, por tanto, de la escala y se toma como valor s. Esta dependencia también está presente en las respuestas onduladas de Haar, tomando

s como referencia, de tal forma que a mayor escala, mayor es la dimensión de las respuestas onduladas.

Tras esto, se utilizan imágenes integrales nuevamente, para proceder al filtrado mediante las máscaras Haar y obtener las respuestas en ambas direcciones. Para obtener la respuesta en la dirección x e y se necesitan únicamente seis operaciones. La longitud de las ondas es 4s.

Una vez que las respuestas han sido calculadas son ponderadas por una gaussiana de $\sigma = 2,5s$ centrada en el punto de interés. Las respuestas son representadas mediante vectores situando las respuestas horizontales y verticales en el eje de abscisas y ordenadas, respectivamente. Finalmente, para obtener la orientación dominante, se calcula la suma de todas las respuestas de una ventana de orientación móvil cubriendo un ángulo de $\frac{\pi}{3}$, tal y como figura en [24]. Tanto la respuesta vertical como la horizontal de la ventana son sumadas, formando un nuevo vector. El vector de mayor longitud es el que se corresponde con la orientación del punto de interés.

Extracción del descriptor

Para la extracción de descriptor, lo primero que se hace es la construcción de una región cuadrada alrededor de cada uno de los puntos de interés y orientada según la orientación calculada. Se comienza con una región de tamaño 20s que se subdivide en regiones de 4x4, dentro de cada una se calcularán las respuestas de Haar de puntos con una separación de muestreo de 5x5 en ambas direcciones. La respuesta Haar será denotada por d_x y d_y, según sea la dirección. Con el fin de otorgarle mayor robustez ante deformaciones geométricas y error de posicionamiento, las respuestas Haar son ponderadas por una gaussiana de $\sigma = 3,3s$ centrada en el punto de interés.

Finalmente, las respuestas son sumadas en cada subregión y conforman un primer conjunto de entrada para el vector de características. Además, para recoger información de la polaridad de los cambios de intensidad, se realiza la suma de los valores absolutos

de las respuestas Haar. De modo que cada subregión tiene como descriptor un vector de v componentes:

$$v = \left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right)$$

y por tanto, englobando las 4x4 subregiones, resulta un descriptor SURF con una longitud de 64 valores para cada uno de los puntos de interés identificados.

3.1.4 Conclusiones

Tanto SURF como SIFT obtienen puntos característicos de una imagen invariantes en escala y orientación, obteniendo un vector por cada punto descriptor.

Entre las diferencias que se encuentran, SIFT guarda la posición, escala y orientación, debido a que para una misma posición es posible encontrar distintos puntos de interés con diferente escala u orientación. Esto hace que SIFT detecte más del doble de puntos que SURF y que el tiempo de procesamiento sea mayor [30].

La ventaja principal de SURF es el reducido coste computacional, ventaja que constituye el criterio que interesa para la elección de uno u otro, puesto que en este proyecto es fundamental que el tiempo de procesamiento y la capacidad de procesamiento sean lo más reducida posible puesto que lo dispositivos móviles poseen capacidades computacionales limitadas.

3.2 Matching

El emparejamiento de puntos de interés o "*matching*" no es más que establecer una relación de correspondencia entre puntos que a priori son iguales, es decir, cumplen una serie de requisitos que los identifican como tal.

El "*matching*" es una técnica fundamental en aplicaciones de Visión Artificial. A partir de ella ha sido posible la reconstrucción de objetos 3D, identificación de objetos o texto en imágenes, reconstrucción de mapas, y todas aquellas aplicaciones que requieran la identificación de formas a través de imágenes capturadas.

3.2.1 Métodos tradicionales

Para resolver el problema de emparejamiento de puntos de interés existen dos técnicas principales, búsqueda por patrón y flujo óptico.

Búsqueda por patrón

La idea de esta técnica no es otra que la de buscar las apariciones de una imagen patrón sobre una imagen mayor. Las apariciones no tienen por qué ser exactas, sino que puede existir un cierto grado de variación con respecto al patrón.

El método más sencillo de búsqueda por patrones es el "template matching" o comparación de plantillas. Dada una imagen A (de tamaño WxH) y P un patrón (wxh), el resultado es una imagen M de tamaño (W-w+1)x(H-h+1) , donde cada píxel de M(x,y) indica la verosimilitud o probabilidad de que el rectángulo [x,y]-[x+w-1,y+h-1] de A contenga el patrón P.

Para obtener M es necesario aplicar un criterio de optimización, en este caso una medida de similitud o diferencia, entre las porciones de las imágenes.

$$M(x,y) = d\{\{A(x,y),...,A(x+w-1,y+h-1)\},\{P(0,0),...,P(w-1,h-1)\}\}$$

Este criterio de optimización puede ser, por ejemplo, la suma de las diferencias al cuadrado.

$$M(x,y) = \sum_{a=0}^{w-1}\sum_{b=0}^{h-1}(P(a,b) - A(x+a,y+b))^2$$

Como se puede observar esta operación matemática es similar a la convolución. Se obtendrá como resultado valores bajos o altos en función de la probabilidad de que el patrón se encuentre en dicha región de la imagen.

Este criterio de optimización no es único, puesto que también se puede utilizar otras medidas de distancias como el producto escalar de patrones centrados, que equivale a la correlación.

La idea es sencilla, basta con encontrar un patrón representativo de la clase de objetos a buscar, aplicar el "template matching" a la imagen para obtener M y buscar los máximos o mínimos locales de M para localizar todas las coincidencias.

Sin embargo, no siempre se obtendrán los resultados deseados, ya que hay que hacer frente a la variabilidad que presenten los objetos a encontrar en la imagen, variabilidad de tamaño y rotación. Para hacer frente a esto, se debe bajar el umbral de coincidencia, teniendo en cuenta que una condición menos restrictiva puede dar lugar a falsos positivos. En la figura 19 se ilustra este efecto, donde se ha aplicado como función de similitud el producto vectorial y como umbral 0,5.

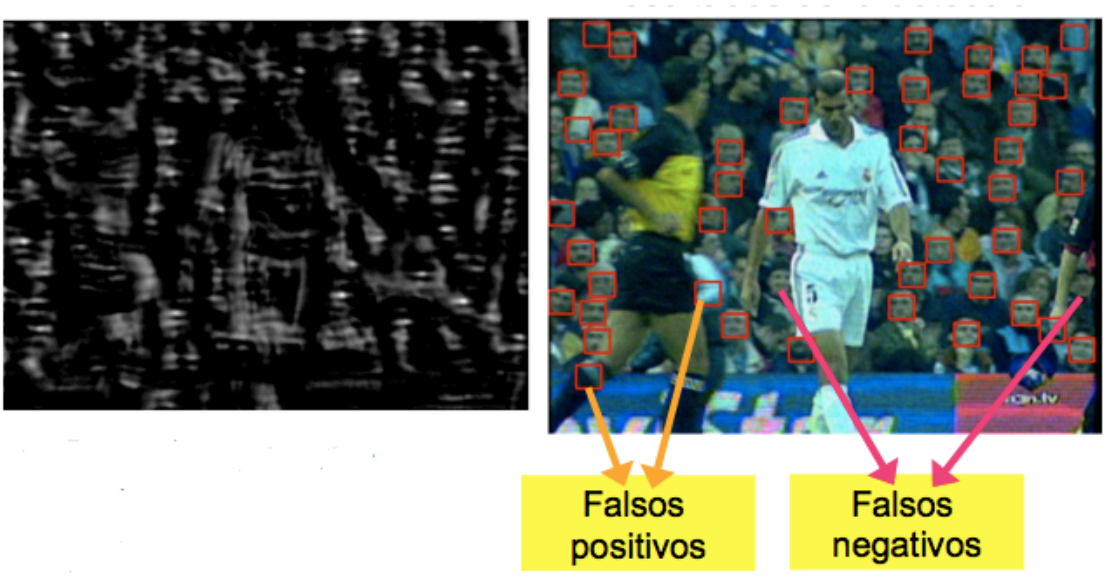

Fig. 19. Mapa de Matching, M (izquierda) y resultados de la detección (derecha)

Esta técnica es por tanto muy sensible a variaciones de rotación, tamaño o cualquier tipo de deformación, y para solucionarlo habría que utilizar varios patrones con distinto

tamaño o rotaciones, hacer una búsqueda multiescala o emplear alguna técnica de atención selectiva, como por ejemplo usar color o bordes para centrar la atención en ciertas partes de la imagen.

Como se mencionó en la introducción del capítulo, estas técnicas permiten llevar a cabo la correspondencia entre puntos. Para ello, se sigue el mismo procedimiento: el patrón se extrae de una imagen y se aplica a otra, siendo el máximo o mínimo matching el que indica la correspondencia entre puntos. En este caso, hay que tener en cuenta que los patrones deben ser muy identificativos.

Fig. 20. Ejemplo de patrones identificativos

En la figura 20 se puede ver que el primer patrón no es nada identificativo, ya que no aporta ninguna característica relevante del entorno que pueda ser utilizada. Los otros dos patrones podrían utilizarse para buscar los puntos característicos en la siguiente imagen pues aportan información característica de los elementos situados en esa región, tal y como se puede ver en la figura 21.

Fig. 21. Resultado de aplicar la búsqueda por patrón al problema de correspondencia entre puntos

Implementación

Para llevar a cabo esta técnica se pueden emplear funciones de la librería OpenCV, entre ellas cvMatchTemplate(), función utilizada en el Estudio Tecnológico que precede a este proyecto.

cvMatchTemplate(const CvArr **image**, const CvArr* **templ**, const CvArr* **result**, int **method**);*

image Imagen donde se correrá la búsqueda. Debe ser de 8-bit o 32-bit floating- point
templ . Patrón buscado; no debe ser mayor que la imagen fuente y debe ser del mismo tipo de datos.

result Mapa de resultado de las comparaciones; monocanal o 32-bit floating-point.
Method Especifica la manera en que el patrón debe ser comparado con las regiones de la imagen.

Los resultados de tiempo de procesamiento obtenidos al utilizar esta función indican que se obtienen mejores tiempos para tamaños de imágenes pequeños y ajustados al tamaño del patrón. Esta función es susceptible a cambios de escala, por lo tanto, si la entrada sobrepasa un cierto tamaño, no será posible reconocer correctamente a que patrón corresponde.

Como conclusión, entre las ventajas de este método se destaca el hecho de que sea una idea sencilla, aunque tiene un gran potencial en detección, reconocimiento y seguimiento de objetos. Entre las desventajas, es un método sensible a rotaciones y escala; es costosa O(WHxh), cuando la resolución aumenta al doble, el tiempo se multiplica por 16.

3.2.2. Flujo óptico

El flujo óptico es una técnica de procesamiento de imágenes que define los vectores de movimiento de diferentes trozos de la imagen. Se aplica a secuencias de vídeo, principalmente. Esto nos permite detectar el movimiento, seguir objetos, y hacer composiciones de vídeo o compresión.

Existen diversas técnicas para calcular el flujo óptico, una de ella es la basada en el "*template matching*", que consiste en dividir la imagen en bloques, y para cada bloque

de una imagen buscar la correspondencia en la otra. Normalmente, no se suelen buscar todos los bloques, puesto que sería demasiado costoso, por lo que teniendo el cuenta el desplazamiento temporal se busca sólo en aquellos con cierta vecindad local.

El flujo óptico, es una técnica más compleja que la anterior puesto que trata con una secuencia de imágenes variantes en el tiempo, por ello, requiere de una serie de parámetros, tales como:

- Tamaño de los bloques a usar. Estos no deben ser ni muy grandes ni muy pequeños, puesto que si son pequeños contienen pocas características y el "*matching*" será poco fiable. Si son muy grandes, se pierde resolución, pocos vectores de movimiento.

-Radio de búsqueda. Determina el tamaño de la zona en la imagen en el instante *t* donde busca el bloque de entrada de la *t-1*. Si es grande, aumentará el tiempo de ejecución. Si es pequeño y el movimiento es mayor, el resultado será impredecible.

- Función de matching a emplear. Para este problema y como ya se vio en el punto anterior, se puede usar una simple función de diferencias o un producto vectorial normalizado, que dota de invarianza frente a cambios de iluminación.

Implementación

A la hora de llevar a cabo la implementación de esta técnica mediante la librería OpenCV, se deberán llevar a cabo los siguientes pasos:

1. Conversión de las imágenes capturadas a escala de grises. Para ello, se creará una imagen en escala de grises a la que se le copiará la información de la imagen original (cvCreateImage (cvSize (IMG_W, IMG_H), IPL_DEPTH_8U,1)).

2. Extracción de características mediante el método seleccionado (SURF, FAST o SIFT).

3. Establecimiento de la correspondencia ente puntos característicos, para ello se tomarán las dos imágenes, la previa y la actual y se comprobará el desplazamiento experimentado para calcular mediante aproximación la distancia recorrida en una

determinada dirección. Como referencia se utilizará el movimiento registrado por el sensor de la cámara.

4. Paso a coordenadas esféricas, de los puntos calculados mediante aproximación de la distancia recorrida.

5. Una vez calculadas las coordenadas de correspondencia en ambas imágenes, construcción de una panorámica, por ejemplo, creando una nueva imagen y solapando las dos.

Un diagrama que refleja de forma más detallada los pasos llevados a cabo para la construcción de la panorámica se muestra en la figura 22.

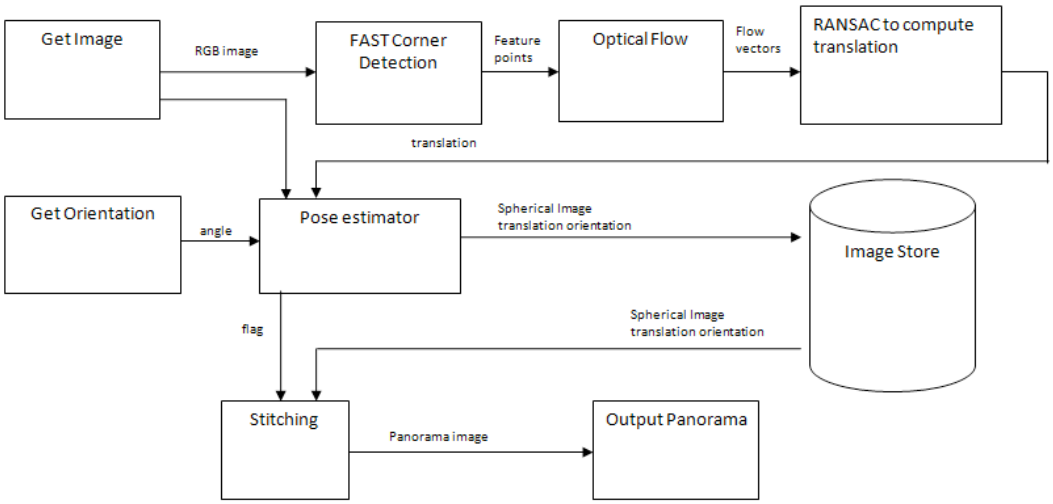

Fig. 22. Construcción de panorámica mediante Optical Flow.

3.2.3. Resolución del problema mediante Teoría de Grafos

La Teoría de Grafos permite resolver una gran cantidad de problemas de diferentes áreas; por ejemplo, en síntesis de circuitos, dibujo computacional, búsqueda de caminos

óptimos, administración y gestión de proyectos, redes sociales o en estudios biológicos y de hábitat, para entender las migraciones.

En este Proyecto Fin de Carrera, el problema a estudiar es el emparejamiento entre distintas imágenes, emparejamiento que tradicionalmente se ha realizado mediante la extracción de características de una imagen y búsqueda de patrones, soluciones que suponen un gran coste computacional, motivación por la cual surge una de las tareas del proyecto: el estudio de "*matching*" mediante la aplicación de Teoría de Grafos y más concretamente mediante la utilización del método de Máximo Cliqué.

El problema del Máximo Cliqué

Un grafo no es más que un conjunto no vacío de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas, orientadas o no. Típicamente un grafo se representa mediante un serie de puntos conectados, en este caso, los puntos serán los obtenidos de los puntos característicos de las imágenes obtenidos mediante SURF.

Un cliqué no es más que un conjunto de vértices. Si son adyacentes dos a dos, se dice que el grafo es completo. El tamaño de un cliqué es el número de vértices que contiene.

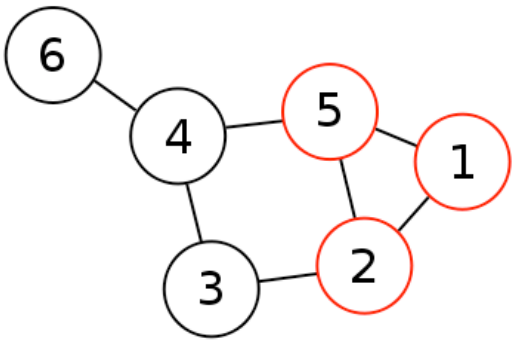

Fig. 23. Ejemplo cliqué.

En figura 23 los vértices 1,2 y 5 forman un cliqué.

El estudio de los cliqués tienen una gran importancia en el campo del procesamiento de señales, temas de visión artificial y principalmente, en problemas de optimización combinacional y en aplicaciones como el análisis de mercados.

Un problema típico relacionado con la teoría de grafos es el problema del máximo cliqué (MCP), que consiste en obtener el subconjunto máximo de vértices tal que sus elementos forman un cliqué. A continuación se muestra un ejemplo (figura 24).

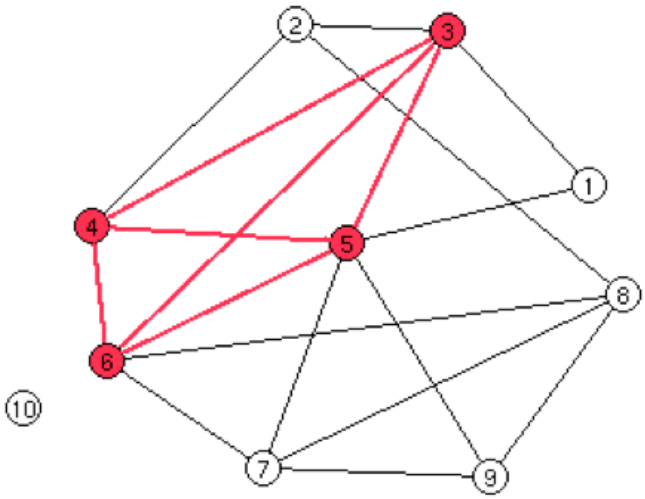

Fig. 24. Ejemplo máximo cliqué.

En este caso se tiene un grafo de 10 nodos, cuyo máximo cliqué tiene 4 nodos, no existe un subconjunto con mayor número de nodos que cumpla que cada uno de ellos esté relacionado con los otros.

El problema del Máximo Cliqué aplicado al tratamiento de imágenes

Un problema que se puede resolver utilizando una variación de Cliqué Máximo es el reconocimiento de segmentos o regiones. Una región no es más que un conjunto de

píxeles que comparten alguna característica. Si se analiza la imagen de la figura 25, según la oscuridad de cada píxel podemos determinar dos regiones bien definidas.

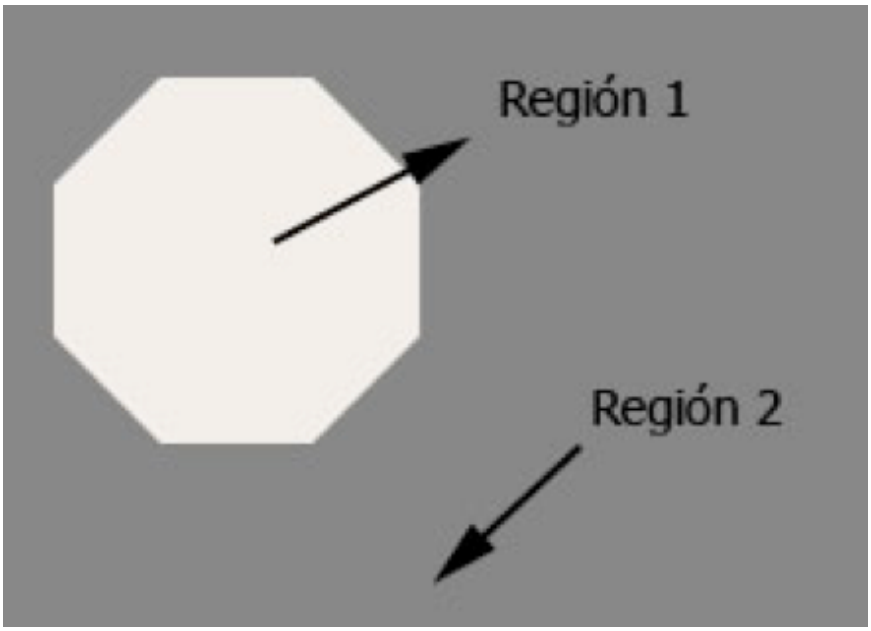

Fig. 25. Ejemplo de distintas regiones a detectar por MCP

El problema de encontrar regiones puede resolverse mediante grafos. Se representa cada píxel de una imagen como un nodo, y la similitud con una arista con un determinado peso. A partir de este modelo, se define un umbral para el algoritmo, de manera que se pueda controlar cómo de similares deben ser dos píxeles para pertenecer a la misma región. Si la similitud entre dos vértices no pasa el umbral, no serán considerados vecinos en el grafo. De esta manera se puede buscar un conjunto de píxeles en la imagen buscando cliqués en el grafo. Si buscamos el máximo cliqué, el resultado será todas las regiones de una imagen.

Aplicación del problema del máximo cliqué al proyecto

Uno de los objetivos de este proyecto es la aplicación del problema del máximo cliqué para enlazar las características comunes de dos imágenes para resolver el problema de búsqueda de patrones.

Para resolver el problema, en primer lugar se extraen las características SURF de las dos imágenes y se calcula el grafo correspondiente para cada una de ellas. Todos los puntos extraídos estarán enlazados entre sí mediante aristas que poseen como peso la distancia euclídea.

Una vez construidos los dos grafos, se construye el grafo de compatibilidad uniendo aquellos nodos cuyas distancias euclídeas son parecidas. Cada nodo representará aquellos dos nodos de características similares.

Una vez construido el grafo de compatibilidad se calcula el máximo cliqué de este nuevo grafo.

Para implementar en JAVA esta funcionalidad se utiliza la librería "JGraph", en la que existen clases como las siguientes:

- UndirectedGraph<V,E>: clase para construir grafo.
- BronKerboschCliqueFinder: Clase que implementa el algoritmo de detección de cliqué de Bron-Kerbosh. Y que posee como método "getBiggestMaximalCliques()" encuentra el máximo cliqué del grafo.

Esta librería hace que aplicar la Teoría de Grafo y en particular el problema de máximo cliqué no suponga una gran complejidad.

Capítulo 4. Análisis y Diseño de la aplicación.

En este capítulo se presentarán los requisitos de la aplicación y el diseño de la misma, mediante diagramas de clase, casos de uso y diagramas de secuencia, con el fin de que el lector pueda tener una visión general del funcionamiento de la aplicación y cómo está diseñada.

4.1 Requisitos

Los requisitos de la aplicación están relacionados con los objetivos que se pretenden alcanzar, es decir, construir una aplicación para estudiar la capacidad de respuesta que presentan los terminales Android al aplicar distintas técnicas de procesamiento de imágenes y la posibilidad de implementar el matching de dos imágenes a través de la información obtenida mediante los descriptores SURF y métodos basados en Teoría de Grafos.

Para abordar este objetivos se implementará una aplicación para Android con dos funciones principales:

- Aplicar sobre una imagen distintas técnicas de procesamiento de imágenes contenidas en la librería OpenCV y estudiar el tiempo de respuesta.
- A partir de dos imágenes aplicar la técnica de extracción de características SURF y establecer la correspondencia mediante Teoría de Grafos, estudiando así su tiempo de respuesta y la viabilidad de utilizar esta técnica para el matching.

El pilar fundamental de la aplicación se basa en la utilización de la biblioteca OpenCV. Esta biblioteca se encuentra en código nativo, por lo que para incluirla en el proyecto es necesario compilarla (tal y como aparece en el Anexo). La compilación de la biblioteca supuso una de las mayores dificultades del proyecto, ya que para ello es necesario instalar diferentes herramientas, las cuales no proporcionaron los resultados deseados bajo el sistema operativo Mac OSX, por lo que se optó por compilar la librería en Windows y después integrarla en el proyecto.

4.2 Diseño

Una descripción sencilla y completa de la aplicación se presenta mediante UML (Unified Modeling Language), estándar para describir de forma gráfica modelos, que permite documentar un sistema, en este caso una aplicación software.

Para describir correctamente la aplicación, se presentan tres de los modelos más comunes y que más información aportan para este tipo de aplicación software: el diagrama de casos de uso, el diagrama de clases y el diagrama de secuencia. También se detalla el diseño de la aplicación y cómo se suministraría.

4.2.1 Descripción. Diagrama de clases

El diagrama de clases de la aplicación final es el que se muestra a continuación:

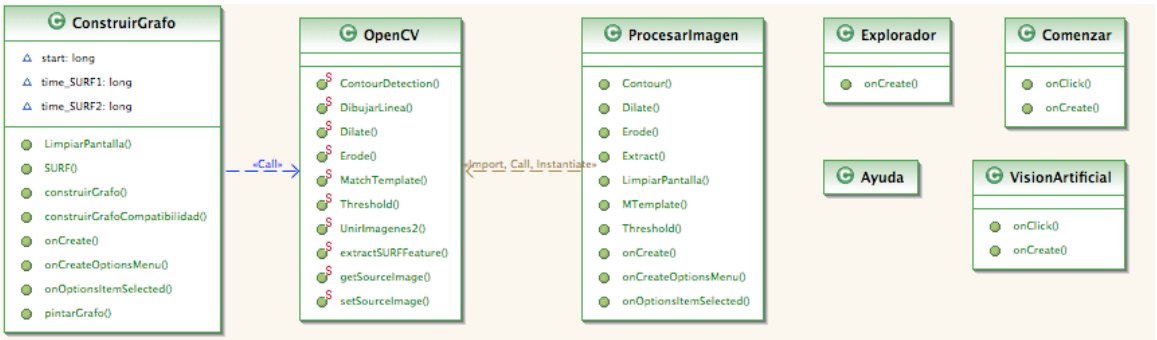

Fig. 26. Diagrama de clases

Como se observa hay dos clases (ConstruirGrafo y ProcesarImagen) que presentan dependencia con la clase OpenCV, clase en la que se cargan los métodos de la librería OpenCV (una vez que está compilada e importada en nuestro proyecto) a través de la siguiente instrucción:

```
System.loadLibrary("opencv");
```

ConstruirGrafo presenta todos aquellos métodos y atributos necesarios para extraer las características SURF de las dos imágenes y construir el grafo de Máximo Cliqué. Se utiliza el método SURF() para extraer las característica SURF y tras este, se invoca al método construirGrafo() para obtener las distancias euclídeas de las características obtenidas. Una vez obtenidas y a través de construirGrafoCompatibilidad(), el cual utiliza funciones de la librería JGraphT, se construye el grafo de Máximo Cliqué empleando las distancias euclídeas.

ProcesarImagen contiene la llamada a distintas funciones de la librería OpenCV. Todos los métodos de esta clase extraen la imagen seleccionada, invocan al método correspondiente de OpenCV y miden el tiempo de procesamiento. Así, por ejemplo, el método Extract() convierte la imagen seleccionada a Bitmap y ejecuta el método extractSURFFeature() de OpenCV.

La clase **VisiónArtificial** muestra la pantalla inicial y mediante el método onClick() ejecuta la acción correspondiente en función del botón pulsado. Así, si se pulsa "Ayuda" se creará una instancia de la clase **Ayuda**, cuya única misión es mostrar un mensaje en pantalla. Si se pulsa "Comenzar", la instancia que se creará será de la clase **Comenzar**, la cual también posee el método onClick() para detectar la selección: "Procesar Imagen".o "Construir Grafo". En cualquier caso se creará una instancia de la clase **Explorador**, a la cual se le pasará por parámetro qué opción ha sido escogida para que pueda instanciar la clase correspondiente (**ProcesarImagen** o **ConstruirGrafo**) tras las selección de la imagen o imágenes.

4.2.2 Diagramas de caso de uso

El diagrama de caso de usos (figura 27) muestra las opciones que el usuario (actor) puede ejecutar en la aplicación.

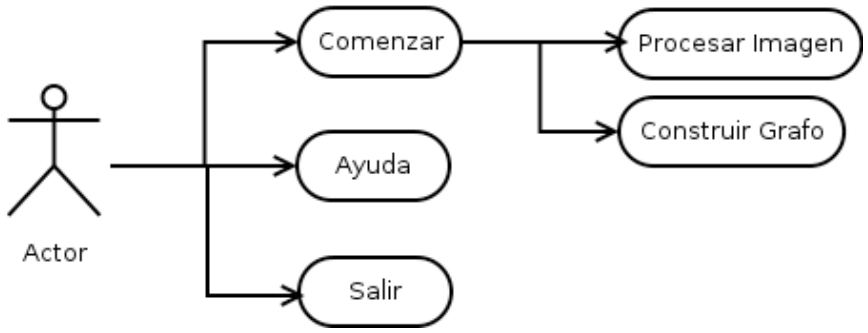

Fig. 27. Diagrama casos de la aplicación

Como se ve en el diagrama de casos de uso existen dos tareas principales que el actor puede ejecutar.

Caso de uso básico 1: Comenzar

La acción que se realiza en este caso de uso es permitir al usuario elegir entre las dos opciones para procesar imágenes que dispone la aplicación. Para ello dispone, a su vez, de dos casos de usos relativos:

Tabla 1. Descripción

Nombre	Descripción
Procesar Imagen	Permite al usuario seleccionar una imagen almacenada en el terminal y realizar funciones básicas de tratamiento de imágenes con el fin de obtener datos del tiempo de respuesta.
Construir Grafo	Permite al usuario extraer las características SURF y/o construir el grafo de máximo cliqué entre dos imágenes, previamente seleccionadas a través de un explorador de archivo, que se muestra tras seleccionar la opción “Construir Grafo”

Caso de uso básico 2: Ayuda

La acción que realiza este caso de uso es únicamente informativa, mostrar un mensaje de ayuda.

Caso de uso básico 3: Salir

La acción que permite este caso es salir de la aplicación.

4.2.2 Diagramas de flujo

Como ya se han mencionado, el sistema implementado consta de dos funcionalidades claramente diferenciadas: el procesamiento de la imagen mediante funciones de OpenCV y la creación del grafo de Máximo Cliqué.

El acceso a cada una de las funcionalidades se realiza mediante un menú donde el usuario decide qué tarea realizar, menú que es accesible a través de una pantalla inicial.

En la siguiente figura podemos el diagrama de flujo general de la aplicación:

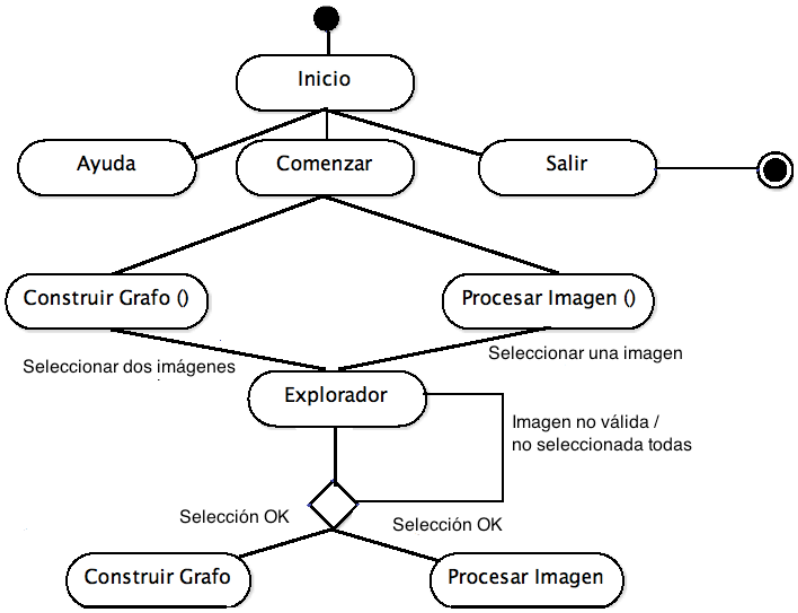

Fig. 28. Diagrama de flujo

Al iniciar la aplicación, se nos presenta una pantalla inicial (figura 29) que nos permite ejecutar las siguientes opciones:

- Ayuda: muestra en pantalla un texto de ayuda para el usuario
- Comenzar: Iniciar la aplicación y acceso a un segundo menú en el que se eligirá que acción ejecutar
- Salir: permite la salida de la aplicación.

Fig. 29. Pantalla inicial de la aplicación

El submenú que encontramos tras pulsar el botón “Comenzar” (figura 30) será el nexo de unión de las dos funciones básicas implementadas: “Construir Grafo ()” y “Procesar Imangen ()”.

Fig. 30. Submenú Comenzar

Tras seleccionar la opción deseada, le aparecerá al usuario un explorador de archivos donde, dependiendo de cual sea la opción de procesamiento, se le solicitará (mediante mensaje) que seleccione una o dos imágenes.

- La opción Procesar Imagen (figura 31) permitirá medir el tiempo de procesamiento al realizar funciones básicas de la librería OpenCV.

Fig. 31. Ejemplo de procesamiento de imagen en la aplicación

- La opción Construir Grafo (figura 32) permitirá medir el tiempo de procesamiento al extraer las características SURF y/o construir el grafo de máximo cliqué.

Fig. 32. Ejemplo de grafo construido en la aplicación

Capítulo 5. Implementación de la aplicación

En este apartado se va a describir de forma detallada los pasos seguidos para implementar la aplicación y las principales funciones utilizadas.

5.1. Implementación de la aplicación Android

Los pasos a seguir para desarrollar la aplicación bajo la plataforma Android son los siguientes:

1. **Obtener las distintas bibliotecas en el código original** para utilizarlas en el proyecto.

   Este punto es fundamental, es el punto de partida del proyecto. Antes de comenzar la implementación se tuvo que estudiar la posibilidad de añadir librerías en otro lenguaje y limitar el tamaño de esta, ya que los dispositivos tienen características limitadas. Para lo que fue necesario estudiar las distintas librerías que se ajustaban a los requisitos del proyectos y decidir cuál de ellas se ajustaba mejor.

   La más importante en el proyecto es la librería OpenCV, biblioteca destinada principalmente para programar funciones de visión artificial y que es todo un referente en este campo. Hoy en día existen multitud de librerías que contienen funciones de OpenCV y que no poseen el mismo tamaño que la

original (característica a considerar cuando se trabaja con dispositivos de memoria limitada). Tras estudiar y probar algunas de ellas se optó por la utilización de la del proyecto de Billmccord.

La librería para la utilización de grafos ya es una librería compilida en JAVA, JGraphT, por lo que incorporarla y utilizarla no supuso mayores complicaciones.

2. **Estudiar cómo poder incluirlas en el proyecto y poder crear funciones a partir de ellas**.

Una vez seleccionada la librería de OpenCV es necesario importarla compilada a nuestro proyecto Android (los pasos seguidos para la compilación se detallan en el Anexo). Una vez importada es necesario crear una clase que nos permita acceder a sus funciones.

La librería JGrpahT la podemos descargar de su web ya compilada y se importa fácilmente al proyecto Android.

En la siguiente figura se muestra la distribución del proyecto Android y dónde se encuentran las librerías mencionadas.

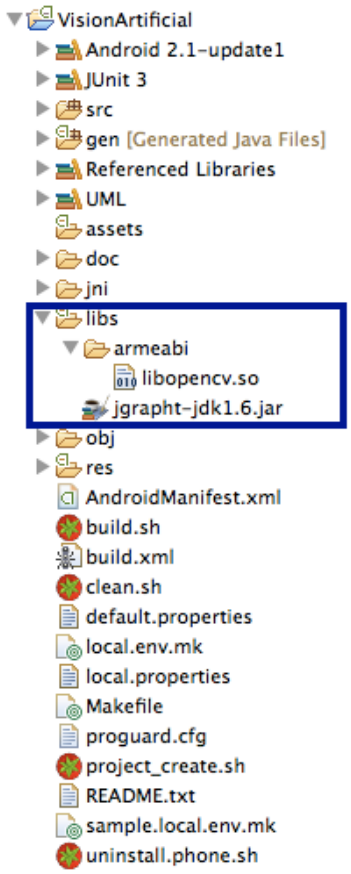

Fig. 33. Estructura del proyecto en Eclipse. Librerías

3. Crear una clase a través de la cual acceder a las funciones de la biblioteca.

Para poder acceder a la librería de OpenCV importada a nuestro proyecto, debemos tener una clase en la que se cargue la referencia a la misma y que nos permite acceder a sus funciones.

Para acceder JGraphT no será necesario compilarla sino incluirla en las propiedades del proyecto e importarla en la clase en la que se utilice sin necesidad de utilizar ninguna referencia.

4. Crear una nueva interfaz gráfica de usuario con la que mostrar los resultados e interactuar con el usuario.

Eclipse nos permite crear fácilmente la interfaz gráfica mediante editores gráficos o mediante la programación en xml (ver figura 34).

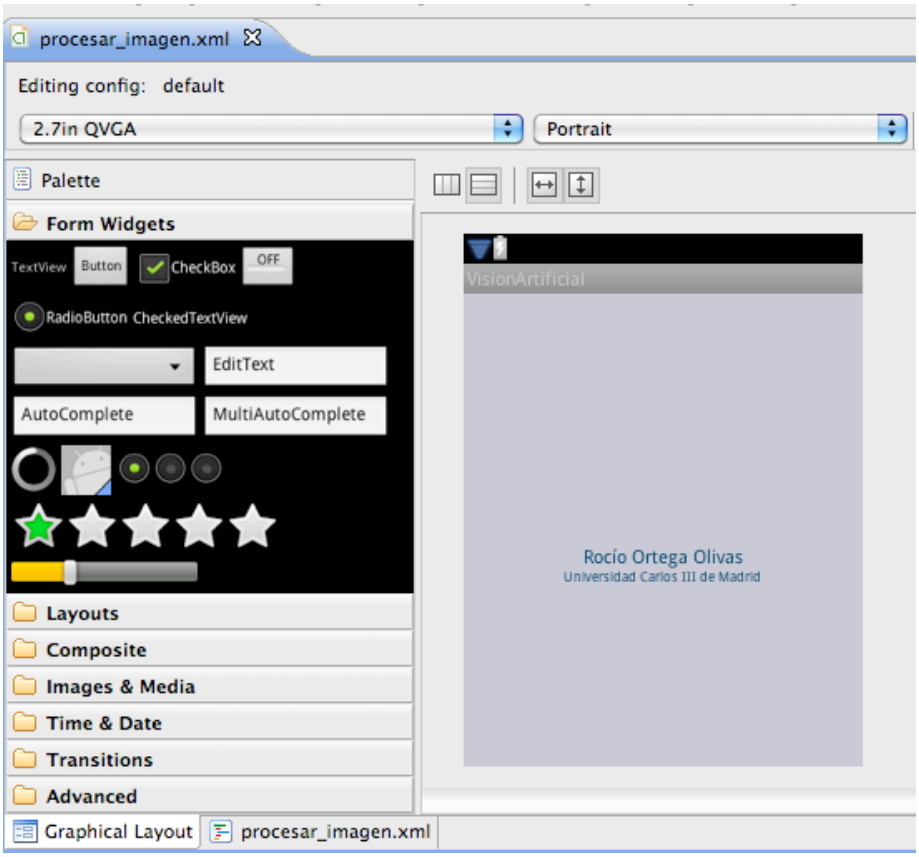

Fig. 34. Creación de interfaz gráfica en Eclipse

Se crearán tantas como pantallas y elementos gráficos sean necesarios. Para el caso de este proyecto, son los que aparecen en la figura 35.

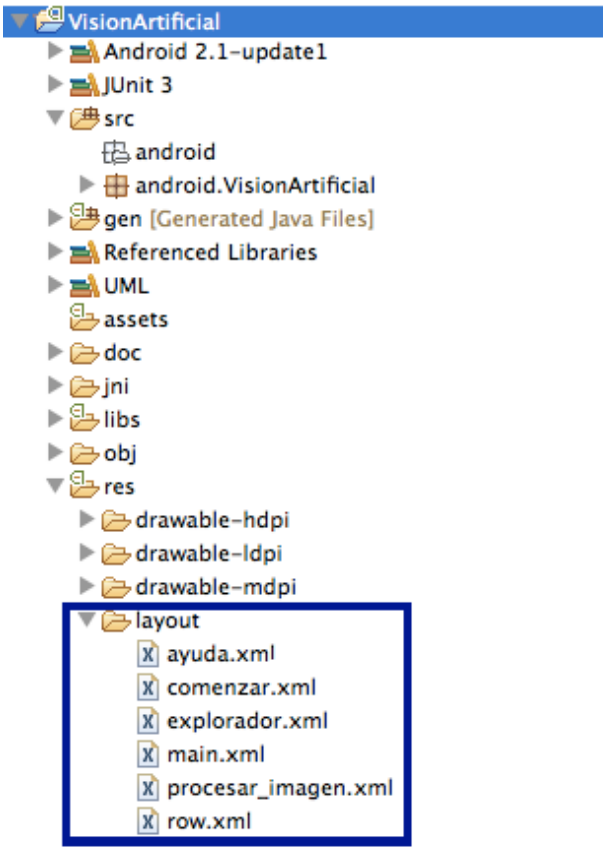

Fig. 35. Elementos gráficos del proyecto en Eclipse

5. **Considerar las limitaciones (según resolución) que existen al mostrar imágenes en las pantallas Android.**

   Hay que tener en cuenta que no todos los terminales tienen las mismas características de resolución, por lo que en el diseño de las pantallas es un dato a tener en cuenta, puesto que se busca la mayor compatibilidad.

6. **Crear las clases necesarias para la transición entre pantallas y para el procesamiento mediante las funciones de OpenCV.**

   Para ello es necesario tener claro la secuencia de actividades que se ejecutarán y qué se debe mostrar en cada momento. Las clases creadas se pueden ver en la siguiente figura.

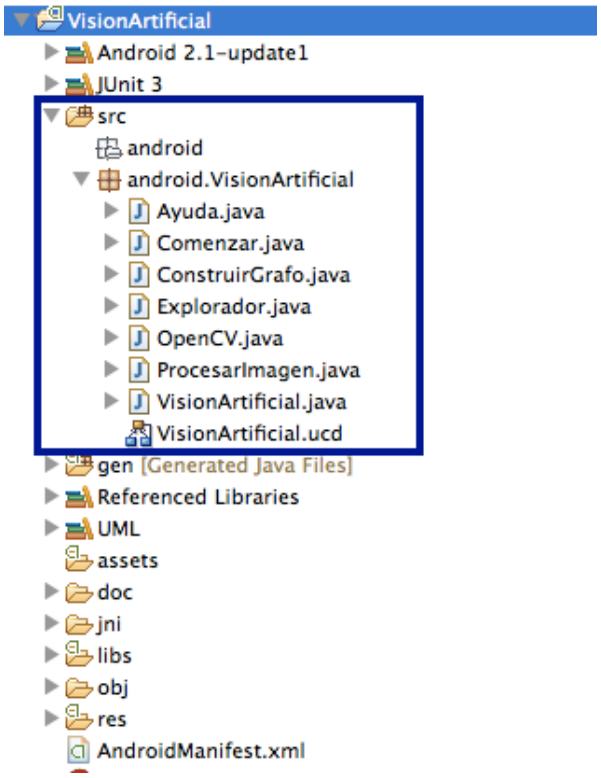

Fig. 36. Clases implementadas en el proyecto

7. **Estudiar cómo implementar el procesamiento basado en Teoría de Grafos y características SURF.**

Una vez extraídas las característica SURF hay que ver cómo se puede almacenarlas para utilizar las funciones importadas de JGrpahT, es decir, hay que estudiar cómo utilizar las funciones que nos permiten el procesamiento basado en Teoría de Grafos con la información que disponemos.

Siguiendo los pasos anteriores, se puede concluir que el desarrollo de la aplicación se divide en los siguientes puntos, los cuales son decisivos para llegar a cumplir el objetivo del proyecto:

1. Creación de la Interfaz Gráfica de Usuario.
2. Creación de las clases para el acceso a la interfaz gráfica y para realizar el procesamiento.
3. Selección de imágenes en el terminal Android mediante el explorador de archivos.

4. Utilización de funciones de la librería OpenCV sobre las imágenes.
5. Extracción de características SURF y *matching* mediante Teoría de Grafos.
6. Configuración AndroidManifest.xml

Interfaz Gráfica de Usuario

Para crear la parte visual de la aplicación que permite la interacción con el usuario, se utilizarán las herramientas que ofrece Eclipse para llevar a cabo dicha tarea y que han sido las responsables de que se haya decido desarrollar el diseño de la GUI siguiendo la estructura habitual de la plataforma Android.

Eclipse permite el diseño de las pantallas mediante el uso de herramientas gráfica y mediante la programación en XML (figura 37).

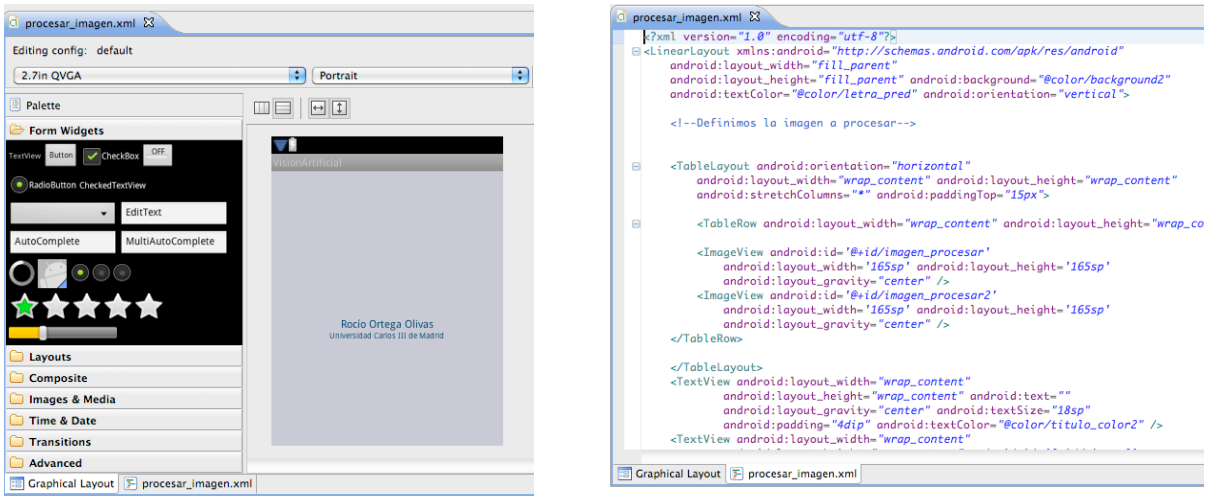

Fig. 37. Creación de la interfaz gráfica en Eclipse

Este proyecto contiene distintas pantallas necesarias para la interacción del usuario, las cuales (figura 38) se describen a continuación.

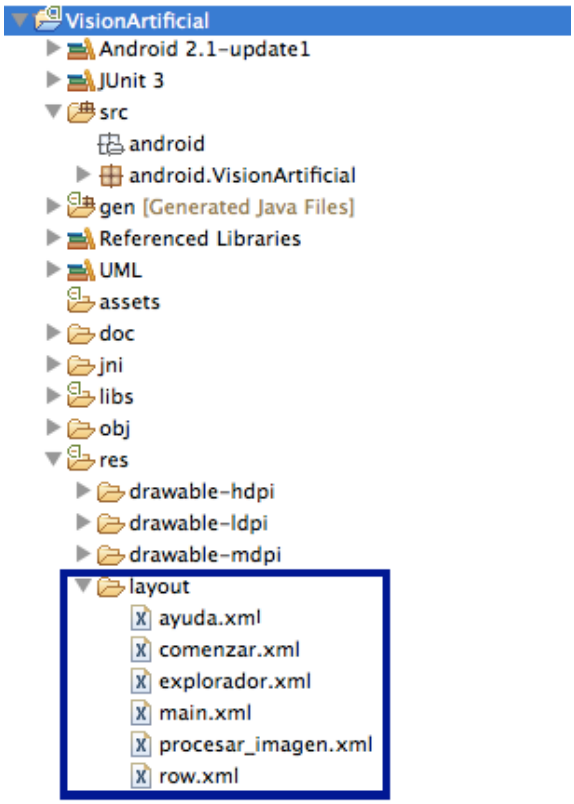

Fig. 38. Interfaz gráfica creada para el proyecto

Tabla 2. Descripción pantallas implementadas

Nombre	Descripción
ayuda.xml	Pantalla para mostrar el texto de ayuda tras pulsar el botón Ayuda de la pantalla principal
comenzar.xml	Pantalla que nos muestra las opciones: Procesar Imagen y Construir Grafo
explorador.xml	Pantalla que nos muestra el explorador de archivos
main.xml	Pantalla que nos muestra la pantalla inicial de la aplicación
procesar_imagen.xml	Pantalla que se utiliza para el procesamiento de la imagen, tanto para Procesar Imagen como para Construir Grafo
row.xml	Elemento gráfico para mostrar en las líneas del explorador de archivos

Clases JAVA implementadas

Para poder acceder a la interfaz gráfica, registrar la intervención del usuario y llevar a cabo todo el procesamiento es necesario crear clases que implementen el código. Este proyecto se compone de las siguientes clases:

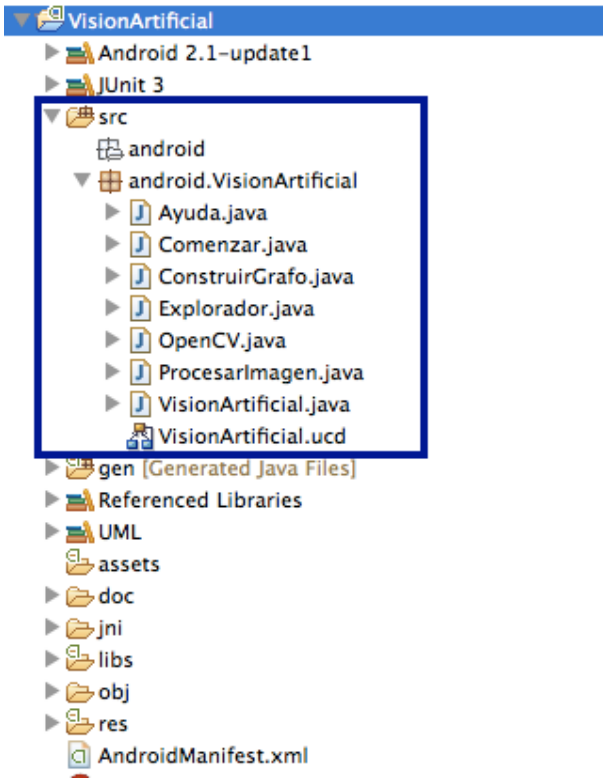

Fig. 39. Clases JAVA del proyecto

Tabla 3. Descripción de las clases implementadas

Nombre	Descripción
Ayuda.java	Clase que muestra la pantalla "Ayuda"
Comenzar.java	Clase que muestra la pantalla "Comenzar" y en función de la opción pulsada por el usuario redirige la aplicación a la acción correspondiente

ConstruirGrafo.java	Clase que ejecuta todo el procesamiento del cálculo del grafo y muestra los resultados en pantalla
Explorador.java	Clase que muestra el explorador y almacena la ruta de las imágenes seleccionadas
OpenCV.java	Clase que importa la librería OpenCV y nos permite utilizar sus funciones
ProcesarImagen.java	Clase en la que se lleva a cabo todo el procesamiento de la imagen utilizando funciones de OpenCV y muestra los resultados
VisionArtificial.java	Clase que muestra la pantalla inicial

Explorador de archivos

Para poder seleccionar dos imágenes en el terminal Android, se optó por la implementación de un explorador de archivos, capaz de obtener la ruta de la imagen seleccionada.

Para su implementación se creó una nueva clase que extiende a la clase ListActivity, y cuyo directorio a mostrar es el directorio raíz del terminal. Cada vez que se selecciona cada uno de los ítems de la lista (directorio en el que nos encontramos) comprobamos si es posible acceder a otro nivel del directorio, es decir, si es una carpeta. En tal caso accedemos y mostramos el contenido. En caso contrario, si es una imagen se sale del explorador y se procesa, y si no es un archivo de imagen, se muestra un mensaje de error indicando que la selección no es válida.

Fig. 40. Explorador de archivos

Utilización de las funciones de la librería OpenCV

Para poder utilizar la gran cantidad de funciones disponibles en la biblioteca, se debe importar compilada al proyecto (Anexo IV) y crear una clase java auxiliar en la que se cargarán las funciones de la librería.

```
public class OpenCV {

	/**
	 * Libreria para cargar los metodos nativos.
	 */
	static {
		System.loadLibrary("opencv");

	}

  /*Métodos que deseamos importar*/
public native static boolean setSourceImage(int[] pixels, int width,
			int height);

	public native static boolean MatchTemplate(int[] pixels, int width,
			int height);

	public native static byte[] getSourceImage();

	public native static float[] extractSURFFeature();
```

```
        public native static void Threshold();
        public native static void ContourDetection();
        public native static void Dilate();
        public native static void Erode();
}
```

De esta forma esta clase podrá ser utilizada en otras del mismo paquete con tan sólo escribir el nombre de la clase y el método, por ejemplo: OpenCV.Threshold();

Extracción de las características SURF y *matching* mediante Teoría de Grafos.

Una vez que se puede acceder a las funciones de la biblioteca OpenCV para extraer las características SURF y hacer el matching de dos imágenes se deben seguir los siguientes pasos:

1. Establecer la imagen a procesar y extraer las características SURF, almacenando el resultado en un array de tipo float para la primera imagen.
2. Establecer la imagen a procesar y extraer las características SURF, almacenando el resultado en un array de tipo float para la segunda imagen.

```
/* Cargamso la imagen */
OpenCV.setSourceImage(pixels, width, height);
/* Extraemos las caractecticas SURF de la imagen */
float [] array = OpenCV.extractSURFFeature();
```

3. Una vez alamacenadas las características en arrays, construir el grafo para las características de cada imagen, descomponer en coordenadas x e y, y calcular la distancia euclídea a todos sus vecinos para cada característica.

4. Una vez construido el grafo para cada una de las imágenes, construir el grafo de compatibilidad, uniendo aquellos puntos cuyas distancias euclídeas son parecidas.

5. Una vez construido el grafo de compatibilidad, calcular el máximo cliqué:

```
/*
 * Una vez que esta creado el grafo de correspondencias buscamos el
 * MC
*/
BronKerboschCliqueFinder<String, DefaultEdge> finder = new
BronKerboschCliqueFinder<String, DefaultEdge>(grafo_c);
Collection<Set<String>> cliques = finder.getBiggestMaximalCliques();
```

Para ello se utiliza la librería JAVA que permite trabajar con grafos:

```
import org.jgrapht.UndirectedGraph;
import org.jgrapht.alg.BronKerboschCliqueFinder;
import org.jgrapht.graph.DefaultEdge;
import org.jgrapht.graph.SimpleGraph;
```

Configuración AndroidManifest.xml

El archivo AndroidManifest.xmlm es un elemento imprescindible en cualquier aplicación Android. Es generado de forma automática por el plug-in de Eclipse. Representa un manifiesto en XML que describe de forma genérica cada uno de los componentes que forman la aplicación. En este archivo también se incluyen permisos y políticas de seguridad que afectan a la aplicación.

Para la aplicación del proyecto ha sido necesario incluir las siguientes directivas:

```
<uses-sdk android:minSdkVersion="8" />
<uses-feature android:glEsVersion="0x00020000" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Las dos primeras aparecen como requisitos para el uso de librerías nativas.

Las dos últimas permiten acceder a la sdcard y escribir en un dispositivo de almacenamiento externo.

5.3. OpenCV y funciones utilizadas

Hoy en día existen distintos proyectos que tienen como principal objetivo portar la biblioteca OpenCV con el fin de optimizarla para Android.

Para la elaboración de la aplicación que se realiza en este proyecto se ha utilizado la del proyecto de Billmccord, la cual puede ser descargada de [6], ya que de todas las encontradas es la que contiene todas las funciones utilizadas en el proyecto . Para reducir el tamaño de la librería se han eliminado todas aquellas que no se utilizan, dejando tan sólo las siguientes funciones:

JNIEXPORT jboolean JNICALL Java_android_VisionArtificial_OpenCV_createSocketCapture(JNIEnv* env, jobject thiz, jstring address_str, jstring port_str, jint width, jint height);

JNIEXPORT void JNICALL android_VisionArtificial_OpenCV_extractSURFFeature(JNIEnv* env, jobject thiz);

JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_releaseSocketCapture(JNIEnv* env, jobject thiz);

JNIEXPORT jboolean JNICALL Java_android_VisionArtificial_OpenCV_grabSourceImageFromCapture(JNIEnv* env, jobject thiz);

JNIEXPORT jbyteArray JNICALL Java_android_VisionArtificial_OpenCV_getSourceImage(JNIEnv* env, jobject thiz);

JNIEXPORT jboolean JNICALL Java_android_VisionArtificial_OpenCV_setSourceImage(JNIEnv* env, jobject thiz, jintArray photo_data, jint width, jint height);

Éstas se corresponden con las funciones definidas en cvjni.hpp e implementadas en cvjni.cpp y que harán las correspondientes llamadas a las funciones de la librería propia de OpenCV con el fin de ejecutar las acciones deseadas en el código Java de la aplicación a desarrollar.

Nuestra aplicación tiene como objetivo realizar operaciones básicas de procesamiento de imágenes con el fin de comprobar los tiempos de procesamiento

requeridos para distintos tamaños de imágenes. Las operaciones para el tratamiento de imágenes implementadas son las siguientes:

JNIEXPORT void JNICALL Java_android_VisionArtifcial_OpenCV_Dilate(JNIEnv* env, jobject thiz);

JNIEXPORT void JNICALL Java_android_VisionArtifcial_OpenCV_Erode(JNIEnv* env, jobject thiz);

JNIEXPORT void JNICALL Java_android_VisionArtifcial_OpenCV_ThresholdGray(JNIEnv* env, jobject thiz);

JNIEXPORT void JNICALL Java_android_VisionArtifcial_OpenCV_ContourDetection(JNIEnv* env, jobject thiz);

JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_MatchTemplate(JNIEnv* env, jobject thiz, jintArray photo_data, jint width, jint height);

Las funciones deberán implementase siguiendo el formato que ya se ha mostrado:

JNIEXPORT *tipo_dato_devuelto* **JNICALL** **Java**_(*paquete_src_Android*)_(*clase_implementa_funciones nativas*)_(*nombre_función*) (parámetros de la función).

Donde JNIEXPORT y JNICALL se corresponden con macros requeridas para la exportación de las funciones a Java.

Cuando se crean las funciones es necesario conocer el paquete del proyecto Android en el que se incluirá el código src y el nombre de la clase que invocará a la librería e implementar como métodos las funciones nativas creadas.

Como parámetros siempre se debe incluir uno de tipo JNIEnv* y otro jobject: JNIEnv* env y jobject thiz. Esto es debido a que en Java se puede declarar cualquier función inexistente y aunque no se ejecute no pasa nada. También se pueden declarar los métodos nativos como static, así no hay que instanciar el objeto, pero en C se sigue recibiendo un puntero 'thiz' (this), es decir en C siempre se reciben los dos primeros parámetros JNIEnv* env y jobject thiz.

En java se pueden declarar parámetros adicionales (que no existen en C) y la función se ejecuta normalmente, pero estos parámetros deben ser los últimos de la lista de parámetros.

Las funciones implementadas y que se utilizarán como métodos nativos en nuestro proyecto de Android son las siguientes:

Dilate (Dilatar)

Esta función implementará la función básica del procesamiento de imágenes que consiste en dilatar la imagen. La dilatación consiste en una operación morfológica, que puede tener como aplicación principal la reducción del ruido o segmentación.

Teóricamente la dilatación consiste en asignar un valor a un píxel en función de sus vecinos. Un píxel tomará el valor 1 en la imagen procesada, si ese mismo píxel o cualquiera de sus vecinos valían 1 en la imagen original. Un ejemplo sería el mostrado en las figura 41a y 41b.

Fig. 41a. Imagen original

Fig. 41b. Imagen tras Dilate

Esta operación se implementará mediante la función "cvDilate" de la libería de OpenCV.

Como muestra la documentación de la función, esta función de OpenCV dilata la imagen original la estructura de elemento especificada que determina el valor de un píxel vecino en función del valor máximo.

```
JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_Dilate(JNIEnv* env, jobject thiz){

    IplImage* dilateImg =NULL;
    dilateImg=cvCloneImage(pImage);
```

```
    cvDilate(pImage,dilateImg,NULL,4);
    pImage=dilateImg;

}
```

En la función creada para tal objetivo se toma la imagen fuente, ya almacenada en pImage (gracias a la utilización de la función setSourceImage), aunque otra alternativa podría haber sido pasarla como parámetro a esta función. Finalmente se devuelve el resultado mediante la llamada a getSourceImage, puesto que hace que el puntero que contiene la información de la imagen a procesar/devolver apunte al resultado (pImage=dilateImg;).

Los parámetros que se le pasan a la función cvDilate serán la imagen fuente y destino. No se le pasará ninguna estructura de elemento que haga referencia a los vecinos ni ningún otro parámetro concreto. Se utilizará en la versión más sencilla.

Erode

Al igual que Dilate, Erode es otra operación morfológica, aunque en este caso, sólo un píxel tomará el valor 1 si ese mismo píxel y todos sus vecinos valían 1 en la imagen original. Se puede ver un ejemplo en las figuras 42a y 42b.

Fig. 42a. Imagen original

Fig. 42b. Imagen tras Erode

Esta operación se implementa mediante la función "cvErode" de la libería de OpenCV.

Al ser esta operación y la de Dilate operaciones morfológicas sencillas y que sólo requieren la llamada a la función de la librería correspondiente, la implementación de

nuestra función para llevar a cabo dicha acción es la misma, cambiando la llamada correspondiente a la función de OpenCV.

```
JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_Erode(JNIEnv* env, jobject thiz){

    IplImage* erodeImg =NULL;
    erodeImg=cvCloneImage(pImage);
    cvErode(pImage,erodeImg,NULL,4);
    pImage=erodeImg;

}
```

Threshold

Consiste en aplicar un nivel fijado de threshold a la imagen a procesar. Es usado principalmente para obtener una imagen binaria de una imagen en escala de grises o para eliminar el ruido (filtro de aquellos píxeles con valores demasiado pequeños o grandes).

OpenCV dispone de una función que permite realizar esta operación, cvThreshold, la cual será la que principalmente se utilice en nuestra función para realizar esta operación.

Uno de los parámetros es el tipo de operación de threshold a aplicar. distinguirse distinguen los siguientes:

CV_THRESH_BINARY

$$dst(x,y) = \begin{cases} \max Value & if \quad src(x,y) > threshold \\ 0 & resto \end{cases}$$

CV_THRESH_BINARY_INV

$$dst(x,y) = \begin{cases} 0 & if \quad src(x,y) > threshold \\ \max Value & resto \end{cases}$$

CV_THRESH_TRUNC

$$dst(x,y) = \begin{cases} threshold & if \quad src(x,y) > threshold \\ src(x,y) & resto \end{cases}$$

CV_THRESH_TOZERO

$$dst(x,y) = \begin{cases} src(x,y) & if \quad src(x,y) > threshold \\ 0 & resto \end{cases}$$

CV_THRESH_TOZERO_INV

$$dst(x,y) = \begin{cases} 0 & if \quad src(x,y) > threshold \\ src(x,y) & resto \end{cases}$$

La comparación gráfica de los distintos tipos de threshold es la siguiente:

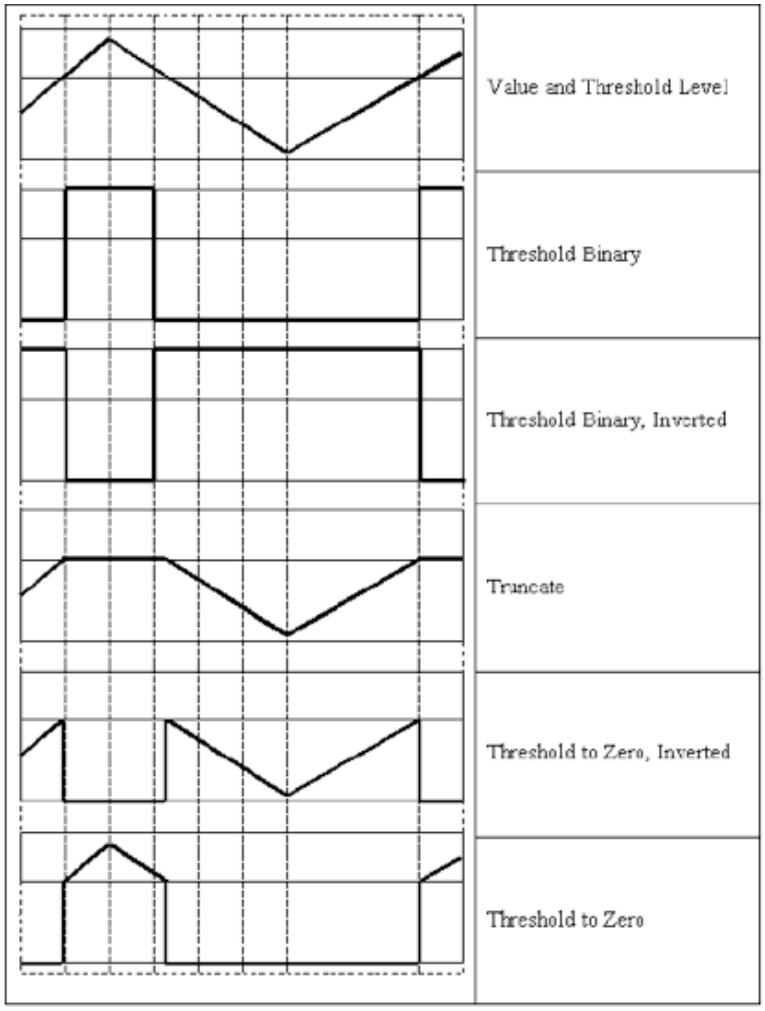

Fig. 43. Comparación de los distintos tipos de Threshold

La función que realizará nuestra operación de threshold es la siguiente:

```
JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_Threshold(JNIEnv* env, jobject thiz){

    IplImage* src;
    IplImage* colorThresh;
    int threshold =120, maxValue=255;
    int thresholdType =CV_THRESH_BINARY;
    src=pImage;
    colorThresh=cvCloneImage(src);
    cvThreshold(src,colorThresh,threshold, maxValue,thresholdType);
    pImage=colorThresh;
}
```

Se ha elegido como tipo el binario, como valor de threshold o de umbral un valor más o menos intermedio en la escala y como valor máximo, el máximo para 8 bits.

La captura de la imagen original y la obtención del resultado se realiza de la misma forma que en el caso de las funciones anteriores. Un ejemplo de esta función sería el mostrado en las figuras 44a y 44b.

Fig. 44a. Imagen original

Fig. 44b. Imagen tras Threshold

Contour_Detection

La identificación del contorno de una imagen puede resultar de gran utilidad para una gran cantidad de aplicaciones.

Esta operación se realizará mediante la función de la librería OpenCV denominada "cvFindContours".

Esta función devuelve el contorno de una imagen binaria (tal y como se muestra en la figura 45b), de ahí que no se obtengan los resultados esperados si se ejecuta sobre otra imagen que no posea estas características.

Fig. 45a. Imagen original

Fig. 45b. Imagen tras Threshold

Se puede seleccionar distintos modos de operación para la localización del contorno por esta función.

Uno de los parámetros de la función es el puntero "first_contour" que devuelve el primer contorno detectado. Si este es NULL se concluye que no se ha encontrado ningún contorno sobre la imagen (por ejemplo en el caso de tener una imagen completamente negra).

Para dibujar el contorno sobre la imagen procesada se usa la función de OpenCV denominada "cvDrawContours".

La función realizada para implementar dicha operación es la siguiente:

```
JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_ContourDetection(JNIEnv* env, jobject thiz){

    IplImage* newImg =NULL;
    IplImage* grayImg =NULL;
    IplImage* contourImg=NULL;

    CvMemStorage * storage=cvCreateMemStorage(0);
    CvSeq * contour=0;
    int mode = CV_RETR_CCOMP;

    newImg=pImage;
    grayImg=cvCreateImage(cvSize(newImg->width, newImg->height),IPL_DEPTH_8U,1);
```

```
    cvCvtColor(newImg,grayImg,CV_BGR2GRAY);
    contourImg=cvCreateImage(cvGetSize(newImg),IPL_DEPTH_8U,3);
    contourImg=cvCloneImage(newImg);

    cvFindContours(grayImg,storage, &contour, sizeof(CvContour), mode,
CV_CHAIN_APPROX_SIMPLE,cvPoint(0,0));

    cvDrawContours(contourImg,contour, CV_RGB(0,255,0), CV_RGB(255,0,0),2,2,8);
    cvReleaseMemStorage(&storage);

    pImage=contourImg;
}
```

Como en casos anteriores toda las acciones a realizar en la función se centran en la llamada a la función de OpenCV correspondiente y descrita anteriormente.

El método de operación seleccionado es CV_CHAIN_APPROX_SIMPLE, en el cual se realiza una compresión horizontal, vertical y diagonal de los segmentos y deja sólo los puntos finales. En cuanto al modo de localización del contorno utiliza CV_RETR_CCOMP, en el cual se localizan todos los contornos y los organiza en dos niveles de jerarquía.

Match_Template

La localización de un patrón en una imagen consiste en ejecutar la operación de correlación entre la imagen a procesar y el patrón que se desea buscar. De esta operación pueden derivar una gran cantidad de aplicaciones.

Esta búsqueda de patrones se realiza mediante la comparación de plantillas (template matching). El procedimiento es el siguiente: *sea una imagen A de tamaño WxH, y P un patrón de tamaño wxh; el resultado es una imagen M, de tamaño (W-w+1)x(H-h+1), donde cada píxel M(x,y) indica la verosimilitud (probabilidad) de que el rectángulo [x,y]=[x+w-1,y-h-1] de A contenga el patrón P.*

La imagen M se define usando alguna función de diferencia o similitud entre dos trozos de la imagen, por ejemplo suma de diferencias al cuadrado, la cual es equivalente a realizar una convolución (pasar una máscara por toda la imagen).

Los valores bajos indican alta probabilidad de que el patrón se encuentre en esa posición, mientras que los valores altos indican probabilidad baja. Para poder decidir cuándo es un valor bajo o alto se debe normalizar.

Por tanto, los pasos a realizar para localizar un patrón son los siguientes:

1. Conseguir un patrón P, representativo de la clase de objetos a buscar.
2. Aplicar template matching a la imagen, obteniendo M.
3. Buscar los máximos y mínimos locales de M.

Un ejemplo sería el mostrado en las siguientes figuras.

Fig. 46a. Imagen original

Fig. 46b. Patrón

Fig. 46c. Resultado

La función de la librería OpenCV que permite realizar esta operación es "cvMatchTemplate", la cual busca el patrón que se le pasa como parámetro, usando la medida de la distancia utilizada en el parámetro "method" de la función.

Esta función posee distintas restricciones:

- Las imágenes fuente y patrón deben ser de un solo canal de 8 bits de profundidad o bien reales de 32 bits.

- La imagen que utilizaremos como patrón debe ser más pequeña que la imagen fuente sobre la que lo buscaremos.
- La imagen resultante, obtenida también como parámetro de la función de be ser de 1 solo canal y necesariamente de reales de 32 bits.
- Si la imagen fuente es de tamaño WxH y el patrón de wxh, la imagen resultante será de tamaño (W-w+1)x(H-h+1)

Para imágenes en color se pueden convertir a escala de grises (cvCvtColor) y trabajar en gris o separar los canales (cvSplit), aplicando el matching a cada canal y sumando los resultados (cvAdd).

Para buscar los máximos y mínimos se utiliza la función cvMinMaxLoc.

Hay distintos modos para localizar el patrón, con CV_TM_SQDIFF el mejor valor de *matching* se encontrará en la posición del mínimo, que será el utilizado en nuestro caso; y con CV_TM_CCORR y CV_TM_CCOEFF el mejor matching será el máximo.

La función implementada sigue los pasos anteriores.

```
JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_MatchTemplate(JNIEnv* env, jobject thiz, jintArray photo_data, jint width, jint height){
IplImage *src, *templ, *ftmp[6];
    IplImage    *img;
    IplImage    *tpl;
    IplImage    *res;
    CvPoint     minloc, maxloc;
    double minval, maxval;
    int img_width, img_height;
    int tpl_width, tpl_height;
    int res_width, res_height;

    img = pImage;

    /*Cargamos la imagen patrón*/
    tpl = getIplImageFromIntArray(env, photo_data, width, height);

    /*Obteneos las propiedades de la imagen*/
    img_width  = img->width;
    img_height = img->height;
    tpl_width  = tpl->width;
```

```
    tpl_height = tpl->height;
    res_width  = img_width - tpl_width + 1;
    res_height = img_height - tpl_height + 1;

    /* Creamos una nueva imagen donde almacenar el resultasdo*/
    res = cvCreateImage( cvSize( res_width, res_height ), IPL_DEPTH_32F, 1 );
       cvMatchTemplate( img, tpl, res, CV_TM_SQDIFF );
    cvMinMaxLoc( res, &minval, &maxval, &minloc, &maxloc, 0 );

    /* Dibujamos un rectágulo rojo para indicar la coincidencia */
    cvRectangle( img, cvPoint( minloc.x, minloc.y ), cvPoint( minloc.x + tpl_width, minloc.y +
tpl_height ),
     cvScalar( 0, 0, 255, 0 ), 1, 0, 0 );
    pImage=img;
}
```

La obtención de la imagen fuente y del resultado se realiza como en casos anteriores.

extractSURFFeature

Función ya implementada en la librería descargada de OpenCV.

Con esta función se realiza la acción propia de la librería de OpenCV cvExtractSURF: extraer los rasgos de una imagen.

Esta función devuelve para cada rasgo su localización, tamaño, orientación y opcionalmente un descriptor.

Un ejemplo de las características extraídas de una imagen sería el siguiente:

Fig. 47. Resultado SURF

Observando la función implementada en cvjni.cpp, se ve que se centra en ejecutar dicha función de OpenCV, es decir, declara las variables que necesitamos para ejecutar la función y extrae todos los puntos que indican rasgos encontrados, marcándolos en la imagen que estamos procesando mediante un círculo (cvCircle).

```
JNIEXPORT void JNICALL Java_android_VisionArtificial_OpenCV_extractSURFFeature(
    JNIEnv* env, jobject thiz) {
    IplImage *pWorkImage=cvCreateImage(cvGetSize(pImage),IPL_DEPTH_8U,1);
    cvCvtColor(pImage,pWorkImage,CV_BGR2GRAY);
    CvMemStorage* storage = cvCreateMemStorage(0);
    CvSeq *imageKeypoints = 0, *imageDescriptors = 0;
    CvSURFParams params = cvSURFParams(2000, 0);
    cvExtractSURF( pWorkImage, 0, &imageKeypoints, &imageDescriptors, storage, params );

    for( int i = 0; i < imageKeypoints->total; i++ )
    {
      CvSURFPoint* r = (CvSURFPoint*)cvGetSeqElem( imageKeypoints, i );
      CvPoint center;
      int radius;
      center.x = cvRound(r->pt.x);
      center.y = cvRound(r->pt.y);
```

```
            radius = cvRound(r->size*1.2/9.*2);
            cvCircle( pImage, center, radius, CV_RGB(255,0,0), 1, CV_AA, 0 );
        }
        cvReleaseImage(&pWorkImage);
        cvReleaseMemStorage(&storage);
}
```

Todas las funciones descritas están implementadas en lenguaje nativo, así como los archivos fuentes y cabeceras descargados de la página del proyecto, y deberán ser compilados para crear la librería .so que será la que utilizaremos en nuestro proyecto de Android. Cualquier modificación en una de las funciones hará necesario volver a compilar la librería.

Capítulo 6. Instalación y ejecución de la aplicación

En este capítulo se explicará cómo instalar y ejecutar la aplicación desarrollada.

6.1. Cómo instalar la aplicación.

Para instalar la aplicación de este proyecto, basta con disponer de un terminal con sistema operativo Android y comprobar que tenga una versión igual o posterior a la versión 2.1.

Para instalarla se debe copiar a nuestro terminal el archivo "VisionArtificial.apk" generado en el directorio "bin" de nuestro proyecto y ejecutarlo, a partir de este momento comenzará la instalación, apareciéndonos la siguiente pantalla en nuestro terminal.

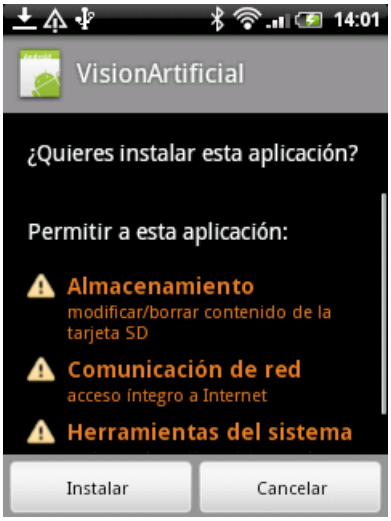

Fig. 48. Instalación aplicación.

Se pulsa “Instalar” y el proceso de instalación comenzará.

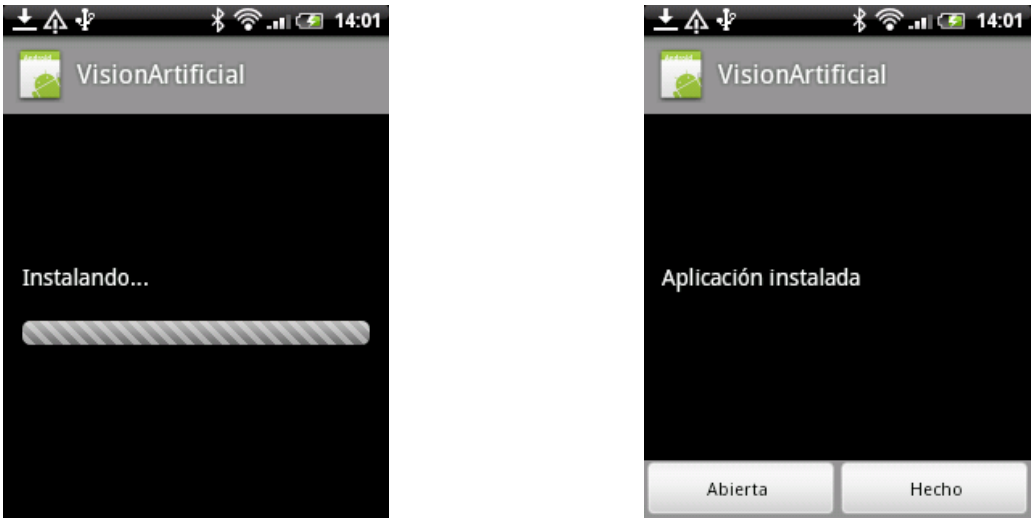

Fig. 49. Instalación aplicación II

La aplicación ya está instalada. Al pulsar “Abierta”, se accede a la aplicación y al pulsar “Hecho” se sale de la instalación.

6.2. Cómo ejecutar la aplicación.

Para ejecutar el programa, es neecsario abrir la aplicación instalada, denominada “VisionArtificial” que aparecerá en el menú de Aplicaciones del terminal Android.

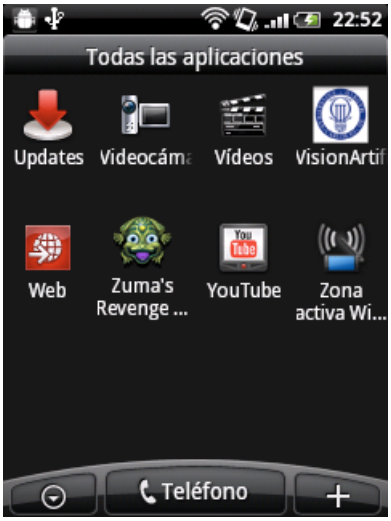

Fig. 50. Icono de la aplicación.

Una vez abierta aparecerá la siguiente pantalla:

Fig. 51. Pantalla inicial aplicación.

Si pulsamos "Ayuda" se mostrará una pantalla con información de ayuda relativa a la aplicación.

Fig. 52. Pantalla ayuda

Al pulsar "Salir" saldremos de la aplicación.

Al pulsar "Comenzar" aparece en pantalla un nuevo menú con las dos acciones que ofrece la aplicación:

- Procesar Imagen. Sobre una imagen ejecutar diferentes funciones de procesamientos de imagen.
- Construir Grafo. Sobre dos imágenes extraer sus características SURF y construir el grafo de máximo cliqué.

Fig. 53. Pantalla tras Comenzar

Si la opción elegida es “Procesar Imagen”, nos aparecerá un explorador de archivos para seleccionar la imagen a procesar

Fig. 54. Pantalla explorador archivos

Una vez seleccionada la imagen a procesar y nos aparecerá la siguiente pantalla.

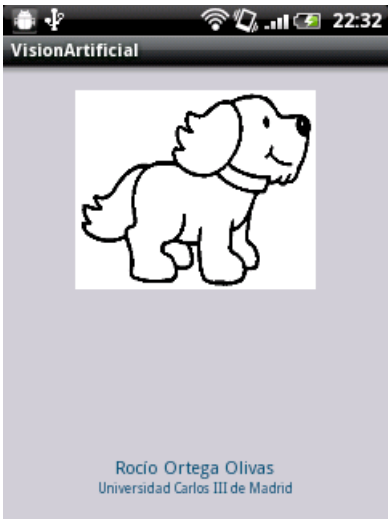

Fig. 55. Pantalla para procesar imagen

En la que veremos la imagen seleccionada, para ejecutar cualquier opción sobre ella, bastará con pulsar “Menú” y aparecerán las distintas opciones de procesamiento.

Fig. 56. Menú

A continuación se muestran los resultados para las acciones anteriores:

Fig. 57a. Dilatar

Fig. 57b. Erosionar

Fig. 57c. Filtrar

Fig. 57d. Detectar contorno

Fig. 57e. Extraer superficie

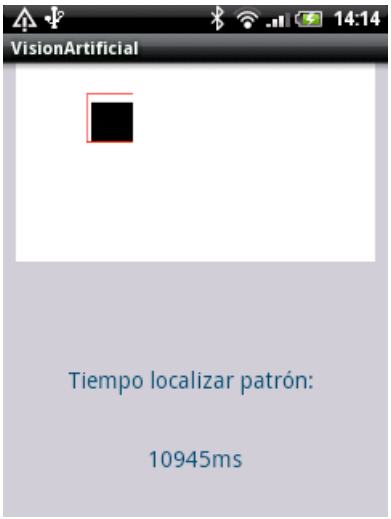

Fig. 57f.Localizar patrón

Si en el explorador seleccionamos una archivo que no es una imagen, aparecerá un mensaje indicándolo, y para seguir con el proceso se tendrá que seleccionar otro que sí lo sea.

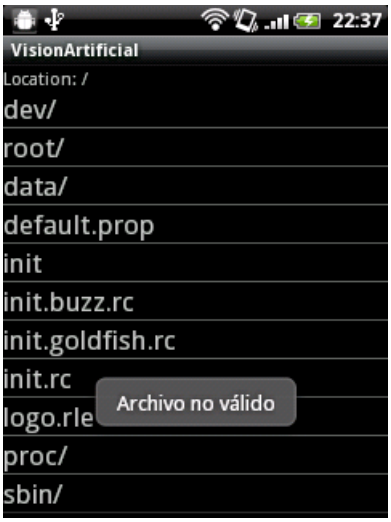

Fig. 58. Archivo no imagen

Si no podemos acceder a un directorio aparecerá el siguiente mensaje de error en pantalla:

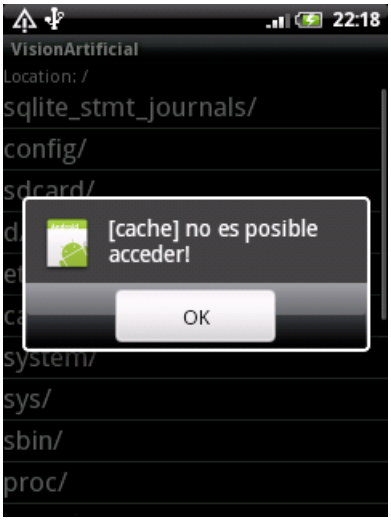

Fig. 59. No acceso a un directorio

Si se selecciona la otra opción, "Construir Grafo":

Fig. 60. Pantalla inicial aplicación.

También aparecerá un explorador, del cual hay que seleccionar las dos imágenes de las que se quiere obtener la correspondencia.

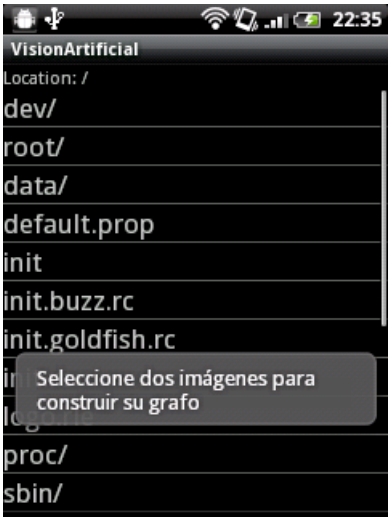

Fig. 61. Pantalla ayuda

Una vez seleccionada la primera imagen a procesar y a continuación de ésta la segunda, aparecerá la siguiente pantalla:

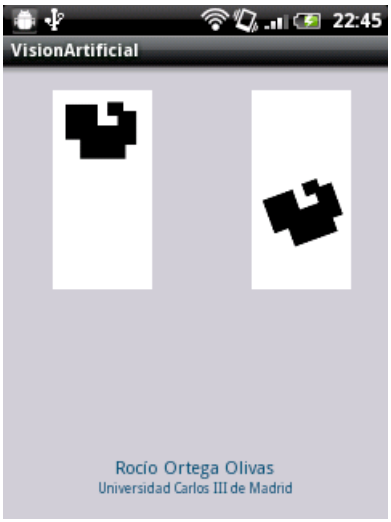

Fig. 62. Procesamiento de imágenes

Al pulsar “Menú” aparecerán dos opciones: SURF y Grafo.

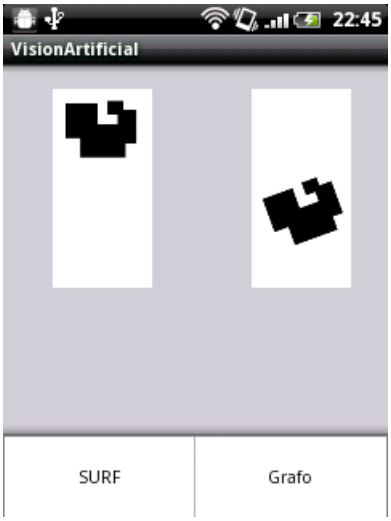

Fig. 63. Procesamiento de imágenes II

Si la opción seleccionada es SURF, para extraer las características, aparecerá en pantalla información de cuánto ha tardado el proceso.

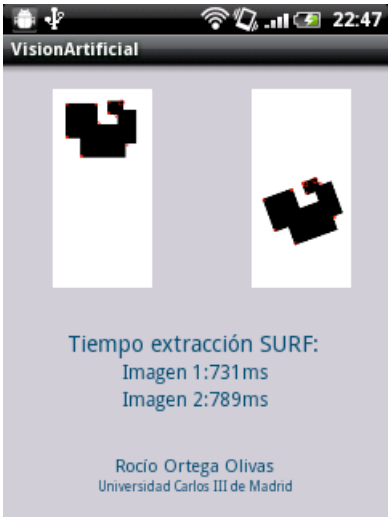

Fig. 64. Procesamiento de imágenes III

Si por el contrario se pulsa Grafo, aparecerá en pantalla la información de lo que ha tardado el proceso y en una única imagen aparecerá trazado el grafo de máximo cliqué.

Fig. 65. Procesamiento de imágenes IV

No es necesario ejecutar primero "SURF" y a continuación "Grafo", el programa está preparado para detectar si ya se han extraído las características de las imágenes o no, en tal caso, al pulsar "Grafo", lo primero que hará será extraerlas.

No siempre es posible encontrar el grafo de máximo cliqué, en tal caso el resultado aparecerá de la siguiente forma:

Fig. 66. Procesamiento de imágenes V. Grafo

Fig. 67. Pantalla grafo no encontrado

Capítulo 7. Resultados experimentales

En este capítulo se mostrarán los resultados experimentales tras la ejecución de la aplicación tanto en la máquina virtual Dalvik como en el terminal.

7. 1 Resultados I: Tiempo de procesamiento

Se estudiarán los tiempos de respuesta al llevar a cabo distintas operaciones de tratamiento de imágenes con el fin de determinar qué características de la imagen proporciona un mejor rendimiento en cuanto a tiempos de respuesta.

7.1.2 Ejecución en el terminal

A continuación se muestra el tiempo de respuesta que presentan las distintas operaciones para el procesamiento de imágenes al ejecutar la aplicación en el terminal Android.

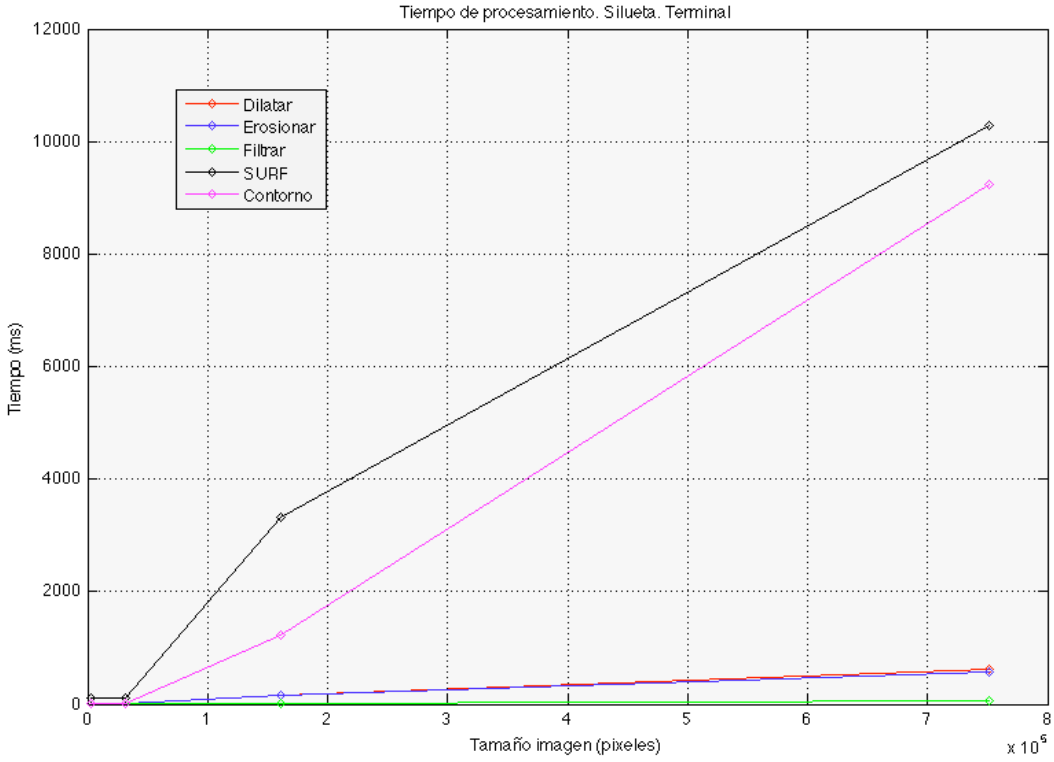

Fig. 68. Tiempo procesamiento silueta en Terminal

En esta primera gráfica está representado el tiempo de procesamiento de las distintas operaciones para el caso de utilizar una imagen con dos niveles de gris, una silueta.

Como se puede observar la acción que requiere un mayor tiempo es la de extraer las característica de la imagen mediante SURF y detectar el contorno, llegando a 10s para imágenes de un tamaño menor que 1Mpixel.

Si en lugar de una imagen con dos niveles de gris se toma una fotografía en color se obtienen los siguientes tiempos de procesamiento mostrados en la figura 69.

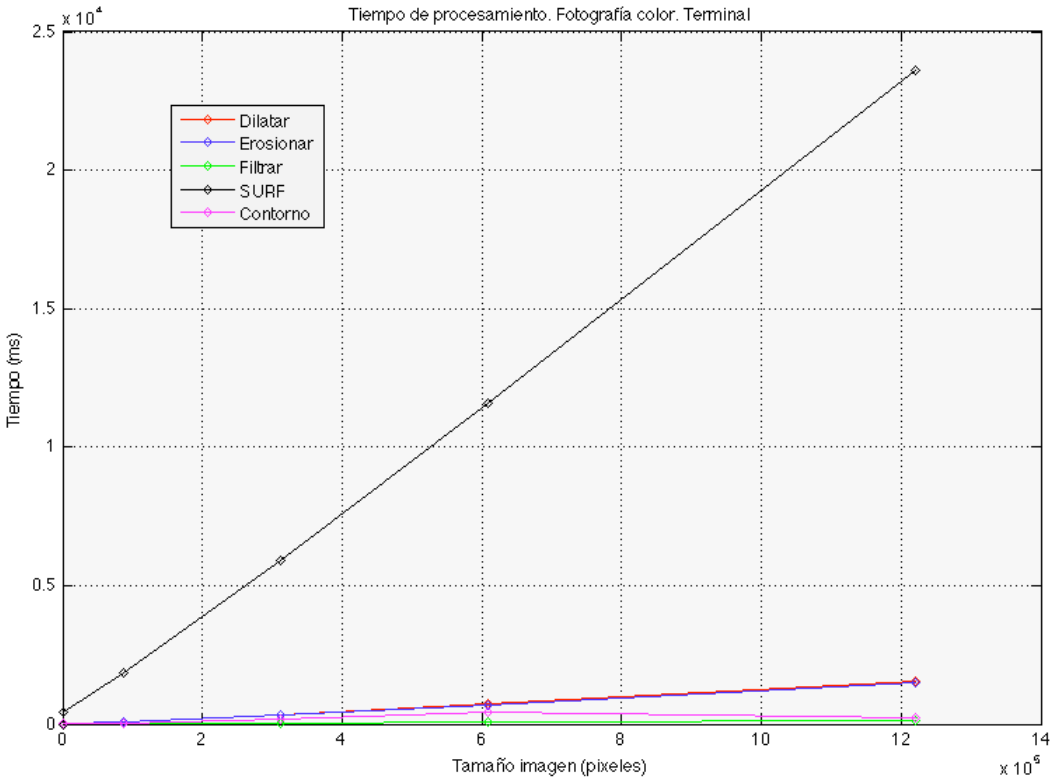

Fig. 69. Tiempo procesamiento fotografía en Terminal

El comportamiento es similar al caso anterior salvo por el hecho de que la operación de contorno presenta una diferencia significativa con respecto al caso anterior, debido a que en este tipo de imágenes el contorno no es detectado.

También se aprecia que el tiempo de procesamiento para imágenes con más de dos niveles de gris es mayor como era de esperar.

Al analizar los resultados obtenidos con imágenes realizadas con la cámara del terminal, se obtiene un comportamiento idéntico al caso anterior (figura 70).

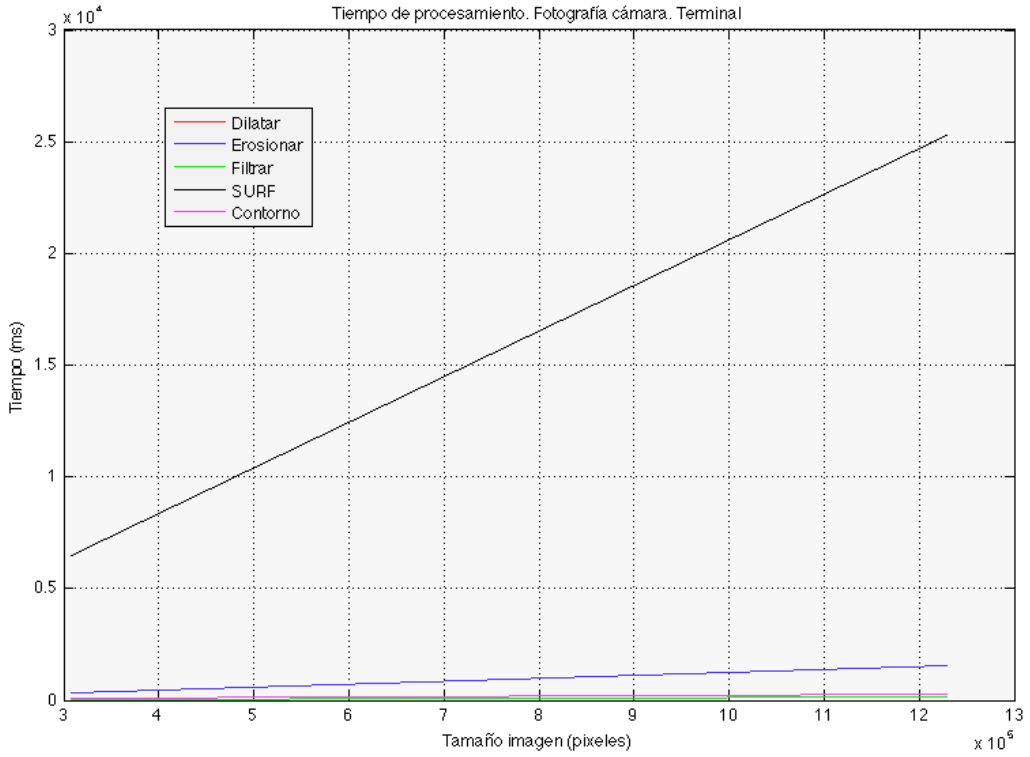

Fig. 70. Tiempo procesamiento fotografía cámara en Terminal

Otra de las operaciones implementadas es la búsqueda de un patrón en una imagen, lo cual requiere la compación de los puntos característicos entre dos imágenes. En la siguiente gráfica se puede observar el tiempo de procesamiento, tanto variando el tamaño del patrón como el de la imagen.

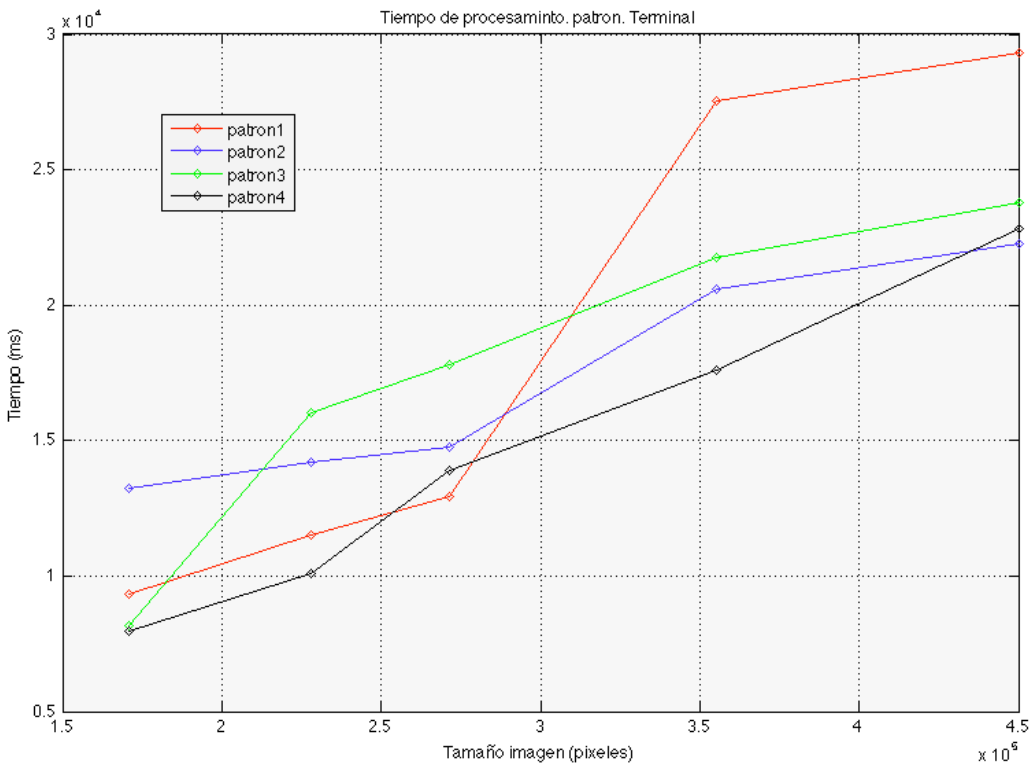

Fig. 71. Tiempo procesamiento búsqueda de patrón en Terminal

Donde los tamaños de patrón son los siguientes:

Patrón1: 170Kp.

Patrón2:227.84 Kp.

Patrón3:236.6 Kp.

Patrón4: 355.20Kp.

Se observa que el tiempo de procesamiento depende tanto del tamaño del patrón como de el tamaño de la imagen. Para patrones pequeños el tiempo es mayor si la imagen en la que buscarlo es grande. De tal forma que la mejor elección sería buscar un tamaño de patrón que se ajuste de forma idéntica al la sección de imagen buscada o uno intermedio.

7.1.3. Ejecución en Dalvik

A continuación (figuras 72 y 73), se muestran los resultados de las operaciones realizadas anteriormente y ejecutadas sobre las mismas imágenes en el simulador. En las gráficas se puede observar que poseen un comportamiento similar, pero el tiempo es mayor que el empleado en el terminal.

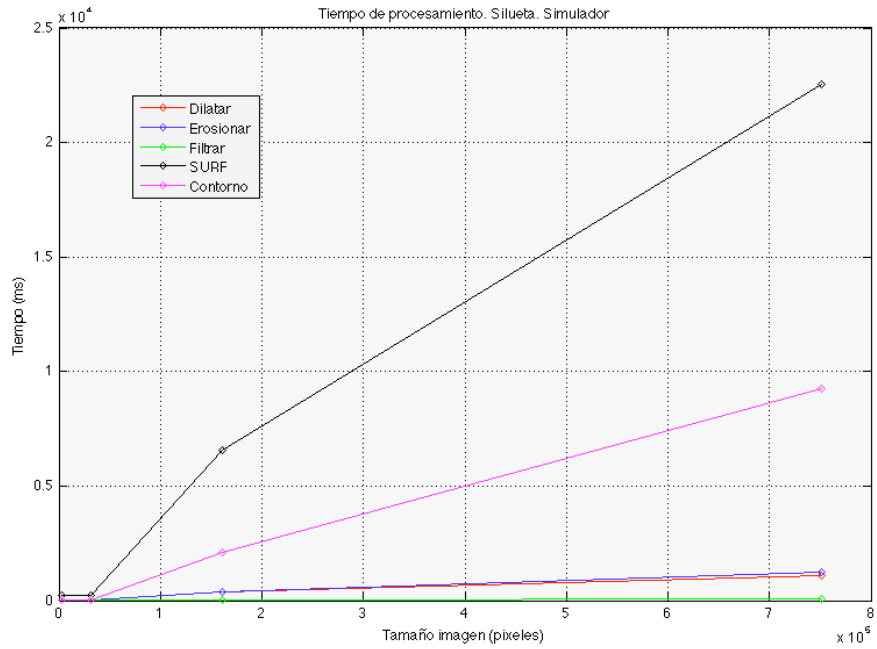

Fig. 72. Tiempo procesamiento silueta en Simulador

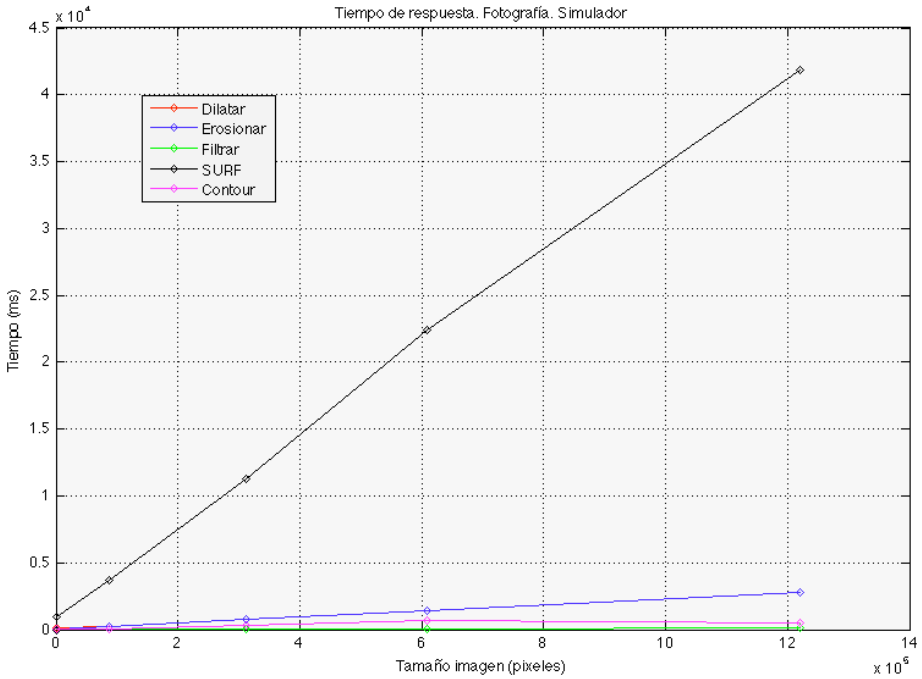

Fig. 73. Tiempo procesamiento fotografía en Simulador

Finalmente, si se comparan los resultados obtenidos al ejecutarlas en la máquina virtual y en el terminal (figuras 74 y 75), se puede verificar que el tiempo es mayor en el simulador y mayor a medida que aumenta el tamaño.

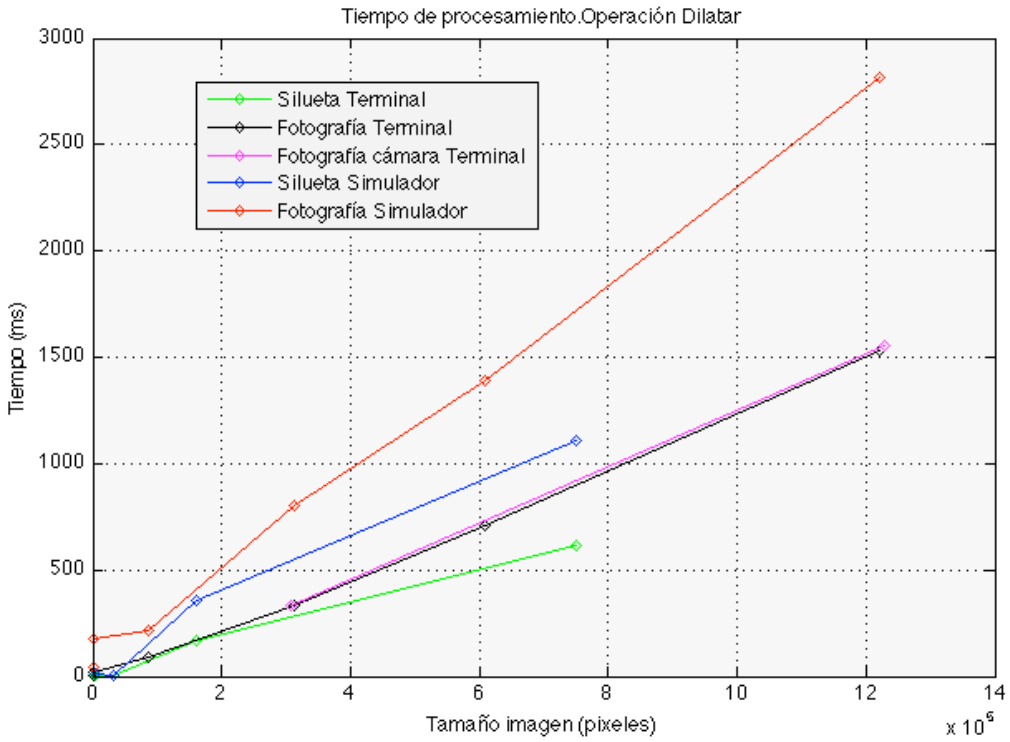

Fig. 74. Comparación operación Dilatar

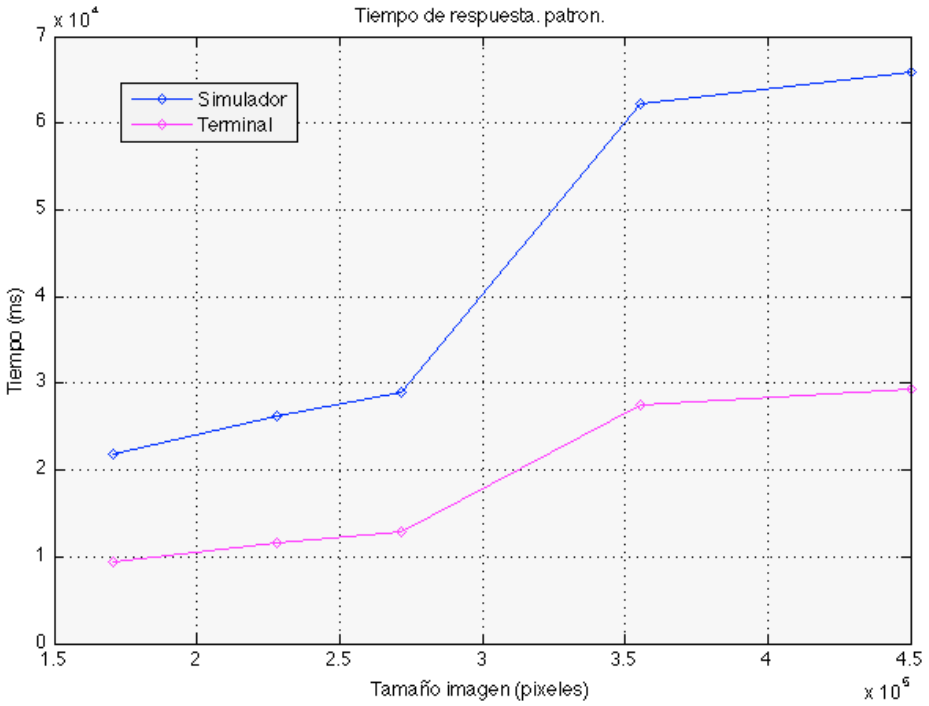

Fig. 75 .Comparación búsqueda de patrón

Los casos de estudio que nos interesan en este proyecto son principalmente la extracción de características mediante SURF y la búsqueda de patrón. Estas dos operaciones suponen mayor tiempo de procesamiento que otras más sencillas presentes en la librería OpenCV, por lo que la elección del tamaño de la imagen es fundamental.

De forma general conviene trabajar con un tamaño de imagen lo más pequeño posible. Durante las pruebas se trató con fotografías tomadas de la propia cámara del terminal con el fin de aproximarse a una situación real en cuanto a tamaño estándar. Se pudo comprobar que el tamaño más pequeño proporcionado funcionaba correctamente, 640x480 píxeles, puesto que proporcionaba unos tiempo de procesamiento adecuados, de tal forma que se puede concluir que es el tamaño óptimo para aplicaciones de tratamiento de imágenes que no tengan exigencias en resolución, pero sí en tiempo de procesamiento.

7.2 Matching mediante MC

Tras hacer diversas pruebas para medir el tiempo de respuesta en un terminal Android de gama media, se comprobó, como era de esperar, que el tiempo total para realizar la correspondencia de características mediante Teoría de Grafo, depende directamente de la extracción de las características SURF, siendo factores determinantes: el número de puntos característicos obtenidos en la extracción que está relacionado con el tipo de imagen a procesar y el tamaño de la imagen, lo cual puede aumentar considerablemente el tiempo de procesamiento.

Imágenes con distintos número de puntos característicos extraídos:

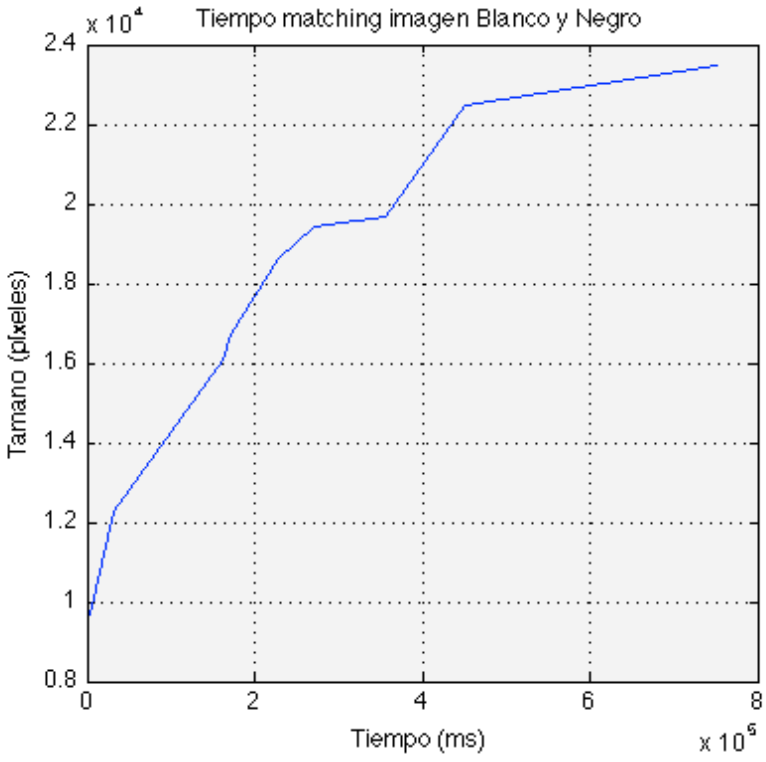

Fig. 76. Tiempo cálculo del grafo de correspondencia. Imagen Blanco y Negro.

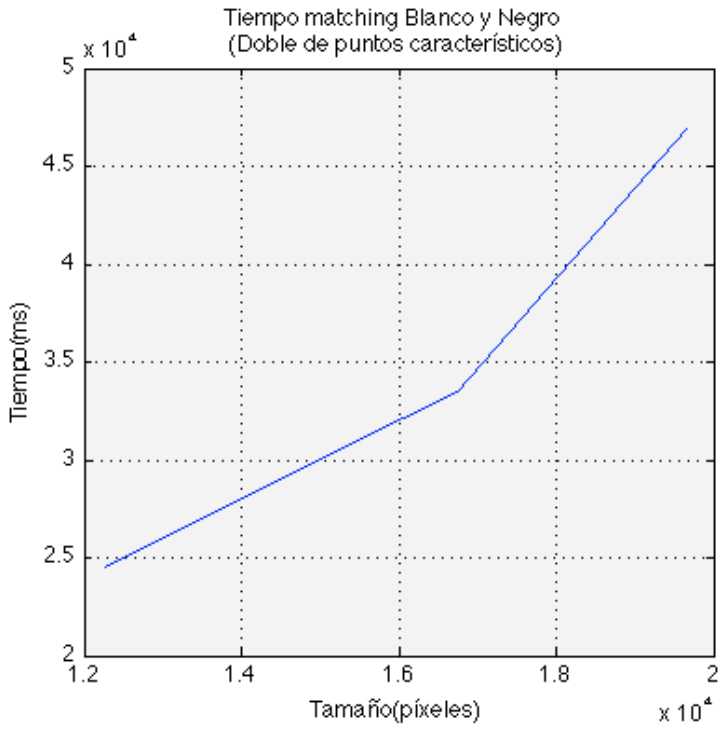

Fig. 77. Tiempo cálculo del grafo de correspondencia. Imagen Blanco y Negro con el doble de puntos.

Imágenes con el mismo número de puntos característicos extraídos:

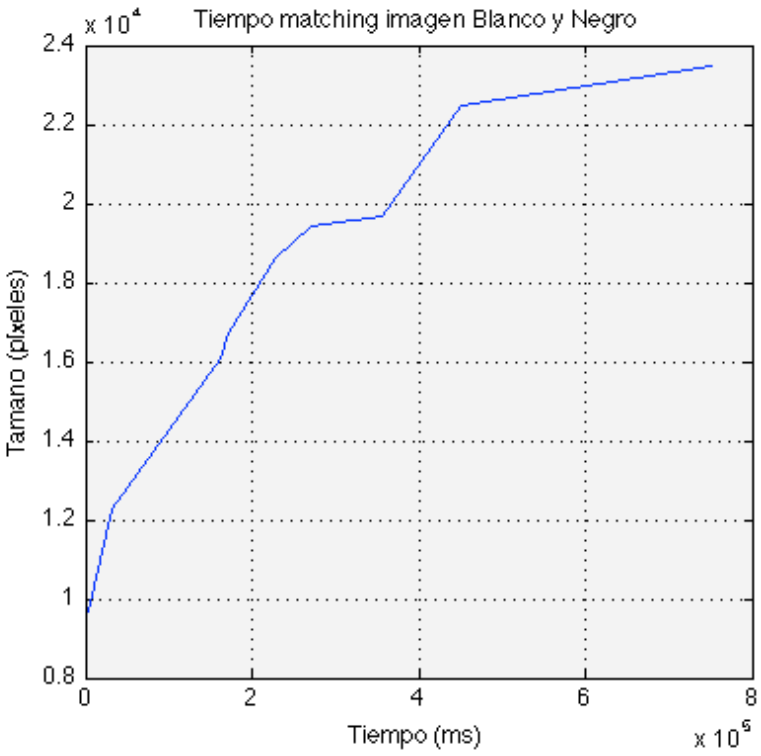

Fig. 78 Tiempo cálculo del grafo de correspondencia. Imagen Blanco y Negro.

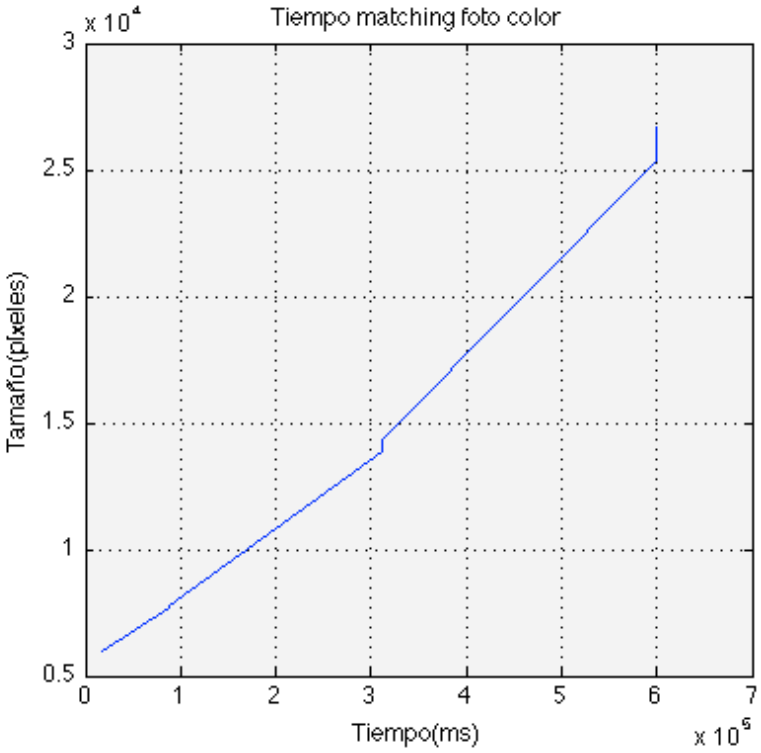

Fig. 79. Tiempo cálculo del grafo de correspondencia. Imagen Color.

Como se puede observar el tiempo de respuesta sigue una tendencia similar, siendo algo mayor para las imágenes en color. En este caso se tomaron imágenes con la misma cantidad de puntos SURF detectados, por lo que la diferencia de tiempo radica principalmente en la extracción de las características SURF.

De tal forma, se puede concluir que el tiempo empleado para calcular el grafo, una vez que las características SURF están extraídas, es despreciable. Es importante tener un número suficiente de descriptores SURF sobre un tamaño de imagen adecuado (no superior a 1 Mpíxel) para evitar tiempos excesivos que pueden llevar a abortar la aplicación.

Capítulo 8. Gestión del proyecto

En este capítulo se explicará cómo se ha gestionado el proyecto, describiendo la metodología de desarrollo empleada, la planificación del proyecto y una estimación de los costes incurridos.

8.1 Introducción

Antes de abordar este capítulo es necesario recordar la definición de "proyecto". Un "proyecto" es un esfuerzo temporal con la finalidad de lograr un único producto, con un comienzo y un final definido, y objetivos específicos que, cuando son logrados significa que se ha completado. Definición proporcionada por el "Project Management Institute".

Para llevar a cabo este apartado se ha empleado la herramienta "Open Project", es un proyecto de open source multiplataforma, que permite gestionar tareas, recursos y hacer un seguimiento visual de todo el proyecto.

8.2 Metodología de desarrollo

Para el desarrollo de este proyecto se ha decidido usar una metodología ágil, más concretamente Scrum, metodología de desarrollo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de evolución del proyecto. Esta metodología tiene como pilares básicos los cuatro puntos de su manifiesto, que se muestra a continuación:

Manifiesto Ágil

"Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros que lo hagan. Con este trabajo hemos llegado a valorar:

- *A los individuos y su interacción, por encima de los procesos y las herramientas.*
- *El software que funciona, por encima de la documentación exhaustiva.*
- *La colaboración con el cliente, por encima de la negociación contractual.*
- *La respuesta al cambio, por encima del seguimiento de un plan.*

Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda."

Esta metodología propone el desarrollo incremental de la funcionalidad del sistema, entregando en cada Sprint (periodo de tiempo entre 2 y 8 semanas) una versión de la aplicación.

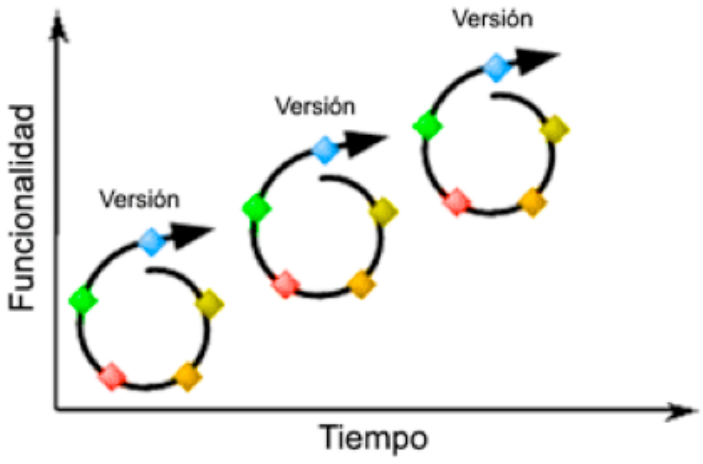

Fig 80. Evolución de la evolución de un proyecto en función del tiempo.

En la figura anterior se puede ver que a diferencia del clásico modelo en espiral, la salida de cada Sprint es teóricamente entregable.

8.3 Organización del proyecto

El desarrollo de este proyecto se ha dividido en dos fases principales, que han quedado distribuidos en el tiempo de la siguiente forma:

- Sprint 1. Implementación de la primera parte de la aplicación. Fecha de finalización 15 de diciembre de 2010.

- Sprint 2. Implementación de la segunda parte de la aplicación. Fecha de finalización 23 de mayo de 2011.

Cabe añadir que además de estas fases se encuentran otras fases que no son de desarrollo como tal, la fase inicial de análisis y la fase de documentación final, por ejemplo, además de fases de pruebas y determinación del diseño. Esto puede verse en los diagramas mostrados posteriormente.

Además, cada diez días aproximadamente se han mantenido reuniones de seguimiento.

	Nombre	Duración	Inicio	Terminado	Predecesores	Nombres del Recurso
1	0 – Inicio del proyecto	0 days	1/10/10 9:00	1/10/10 9:00		
2	**ANÁLISIS**	12,875 d...	**1/10/10 9:00**	**19/10/10 17:00**		
3	Viabilidad y estado del arte del sistema planteado	6,875 days	1/10/10 9:00	11/10/10 17:00		Jefe de proyecto[50%];...
4	1 – Especificación de requisitos	0 days	1/10/10 9:00	1/10/10 9:00		
5	Listado de objetivos	8,875 days	1/10/10 9:00	13/10/10 17:00		Analista
6	2 – Análisis de alternativas y fin de la especificación	1,875 days	14/10/10 9:00	15/10/10 17:00		
7	Estudio de herramientas y alternativas de programación	2 days	16/10/10 8:00	19/10/10 17:00	5	Analista
8	**DISEÑO**	4,875 days	**20/10/10 9:00**	**26/10/10 17:00**		
9	3 – Reunión: Diseño	0 days	20/10/10 9:00	20/10/10 9:00		
10	Seleccionar la estrategia de diseño	1,875 days	20/10/10 9:00	21/10/10 17:00		Analista[50%];Jefe de p...
11	Descomposición de los módulos de funcionalidades	1 day	23/10/10 8:00	25/10/10 17:00	10	Analista
12	Diseño de la aplicación	1 day	26/10/10 8:00	26/10/10 17:00	11	Analista
13	**IMPLEMENTACIÓN PROTÓTIPO**	148,75 d...	**27/10/10 9:00**	**23/05/11 16:00**		
14	4 – Elaboración de software	0 days	27/10/10 9:00	27/10/10 9:00		
15	Implementación aplicación primera parte	24,875 d...	27/10/10 9:00	30/11/10 17:00	12	Programador
16	5 – Fin prototipo inicial	0 days	27/10/10 9:00	27/10/10 9:00		
17	Pruebas y depuración	6 days	1/12/10 8:00	8/12/10 17:00	15	Tester
18	6 – Entrega. Captación de sugerencias, especificación c	0,875 days	8/12/10 9:00	8/12/10 17:00		
19	Reajustes del código	2 days	9/12/10 8:00	10/12/10 17:00	17	Jefe de proyecto[50%];...
20	Memoria explicativa y análisis de resultados	3 days	13/12/10 8:00	15/12/10 17:00	19	Programador
21	7 – Ampliación. Implementación segunda parte	0 days	16/12/10 9:00	16/12/10 9:00	20	
22	Implementación aplicación	65,875 d...	16/12/10 9:00	17/03/11 17:00		
23	Pruebas y depuración	22 days	18/03/11 9:00	19/04/11 9:00		Tester
24	8 – Entrega. Captación de sugerencias	0 days	19/04/11 9:00	19/04/11 9:00		
25	Reajustes del código	14,875 d...	19/04/11 9:00	9/05/11 17:00	23	Jefe de proyecto[50%];...
26	9 – Fin reajustes	0 days	10/05/11 9:00	10/05/11 9:00		
27	Pruebas y depuración	9,875 days	10/05/11 8:00	23/05/11 16:00	25	Tester
28	**DEPURACIÓN Y PLAN DE PRUEBAS**	6,875 days	**25/05/11 9:00**	**2/06/11 17:00**		
29	10 – Inicio pruebas, revisión y depuración del sistema t	2,875 days	25/05/11 9:00	27/05/11 17:00		
30	Retoques del programa total	3 days	28/05/11 9:00	1/06/11 17:00	27	Jefe de proyecto[50%];...
31	9 – Inicio pruebas	0 days	2/06/11 9:00	2/06/11 9:00		
32	Aplicación de las pruebas	0,875 days	2/06/11 9:00	2/06/11 17:00	30	Analista
33	**DOCUMENTACIÓN**	44,125 d...	**6/06/11 8:00**	**5/08/11 9:00**		
34	Preparación de la documentación del proyecto	21,875 d...	6/06/11 8:00	5/07/11 16:00		Analista
35	Revisión del proyecto	15 days	6/07/11 16:00	27/07/11 16:00	32;34	Jefe de proyecto
36	11 – Reunión : Presentación resultado y documentación	0 days	28/07/11 9:00	28/07/11 9:00		
37	Informe final y manual de usuario	4,875 days	29/07/11 9:00	4/08/11 17:00	35	Jefe de proyecto
38	12 – Fin proyecto	0 days	5/08/11 9:00	5/08/11 9:00		

Fig. 81. Distribución temporal del proyecto

Así, tras analizar los recursos utilizados en el desarrollo de la aplicación se presenta el tiempo dedicado a cada fase (figura 82), así como un gráfico porcentual (figura 83).

	Nombre	Trabajo	Duración	Inicio
1	0 - Inicio del proyecto	0 horas	0 days	1/10/10 9:00
2	**⊟ANÁLISIS**	157 horas	12,875 d...	**1/10/10 9:00**
3	Viabilidad y estado de	55 horas	6,875 days	1/10/10 9:00
	Jefe de proyecto	*27,5 horas*	*6,875 days*	*1/10/10 9:00*
	Analista	*27,5 horas*	*6,875 days*	*1/10/10 9:00*
4	1 - Especificación de	0 horas	0 days	1/10/10 9:00
5	Listado de objetivos	71 horas	8,875 days	1/10/10 9:00
	Analista	*71 horas*	*8,875 days*	*1/10/10 9:00*
6	2 - Análisis de alterna	15 horas	1,875 days	14/10/10 9:00
7	Estudio de herramient	16 horas	2 days	16/10/10 8:00
	Analista	*16 horas*	*2 days*	*16/10/10 8:00*
8	**⊟DISEÑO**	31 horas	4,875 days	**20/10/10 9:00**
9	3 - Reunión: Diseño	0 horas	0 days	20/10/10 9:00
10	Seleccionar la estrateg	15 horas	1,875 days	20/10/10 9:00
	Analista	*7,5 horas*	*1,875 days*	*20/10/10 9:00*
	Jefe de proyecto	*7,5 horas*	*1,875 days*	*20/10/10 9:00*
11	Descomposición de los	8 horas	1 day	23/10/10 8:00
	Analista	*8 horas*	*1 day*	*23/10/10 8:00*
12	Diseño de la aplicació	8 horas	1 day	26/10/10 8:00
	Analista	*8 horas*	*1 day*	*26/10/10 8:00*
13	**⊟IMPLEMENTACIÓN PRO**	1.195 hor...	148,75 d...	**27/10/10 9:00**
14	4 - Elaboración de sof	0 horas	0 days	27/10/10 9:00
15	Implementación aplica	199 horas	24,875 d...	27/10/10 9:00
	Programador	*199 horas*	*24,875 d...*	*27/10/10 9:00*
16	5 - Fin prototipo inicia	0 horas	0 days	27/10/10 9:00
17	Pruebas y depuración	48 horas	6 days	1/12/10 8:00
	Tester	*48 horas*	*6 days*	*1/12/10 8:00*
18	6 - Entrega. Captación	7 horas	0,875 days	8/12/10 9:00
19	Reajustes del código	16 horas	2 days	9/12/10 8:00
	Jefe de proyecto	*8 horas*	*2 days*	*9/12/10 8:00*
	Programador	*8 horas*	*2 days*	*9/12/10 8:00*
20	Memoria explicativa y	24 horas	3 days	13/12/10 8:00

20	Memoria explicativa y	24 horas	3 days	13/12/10 8:00
	Programador	*24 horas*	*3 days*	*13/12/10 8:00*
21	7 - Ampliación. Imple	0 horas	0 days	16/12/10 9:00
22	Implementación aplica	527 horas	65,875 d...	16/12/10 9:00
23	Pruebas y depuración	176 horas	22 days	18/03/11 9:00
	Tester	*176 horas*	*22 days*	*18/03/11 9:00*
24	8 - Entrega. Captación	0 horas	0 days	19/04/11 9:00
25	Reajustes del código	119 horas	14,875 d...	19/04/11 9:00
	Programador	*59,5 horas*	*14,875 d...*	*19/04/11 9:00*
	Jefe de proyecto	*59,5 horas*	*14,875 d...*	*19/04/11 9:00*
26	9 - Fin reajustes	0 horas	0 days	10/05/11 9:00
27	Pruebas y depuración	79 horas	9,875 days	10/05/11 8:00
	Tester	*79 horas*	*9,875 days*	*10/05/11 8:00*
28	**⊟DEPURACIÓN Y PLAN I**	54 horas	6,875 days	**25/05/11 9:00**
29	10 - Inicio pruebas, re	23 horas	2,875 days	25/05/11 9:00
30	Retoques del program	24 horas	3 days	28/05/11 9:00
	Jefe de proyecto	*12 horas*	*3 days*	*28/05/11 9:00*
	Programador	*12 horas*	*3 days*	*28/05/11 9:00*
31	9 - Inicio pruebas	0 horas	0 days	2/06/11 9:00
32	Aplicación de las prue	7 horas	0,875 days	2/06/11 9:00
	Analista	*7 horas*	*0,875 days*	*2/06/11 9:00*
33	**⊟DOCUMENTACIÓN**	334 horas	44,125 d...	**6/06/11 8:00**
34	Preparación de la doc	175 horas	21,875 d...	6/06/11 8:00
	Analista	*175 horas*	*21,875 d...*	*6/06/11 8:00*
35	Revisión del proyecto	120 horas	15 days	6/07/11 16:00
	Jefe de proyecto	*120 horas*	*15 days*	*6/07/11 16:00*
36	11 - Reunión : Presen	0 horas	0 days	28/07/11 9:00
37	Informe final y manua	39 horas	4,875 days	29/07/11 9:00
	Jefe de proyecto	*39 horas*	*4,875 days*	*29/07/11 9:00*
38	12 - Fin proyecto	0 horas	0 days	5/08/11 9:00

Fig 82. Distribución temporal del proyecto II.

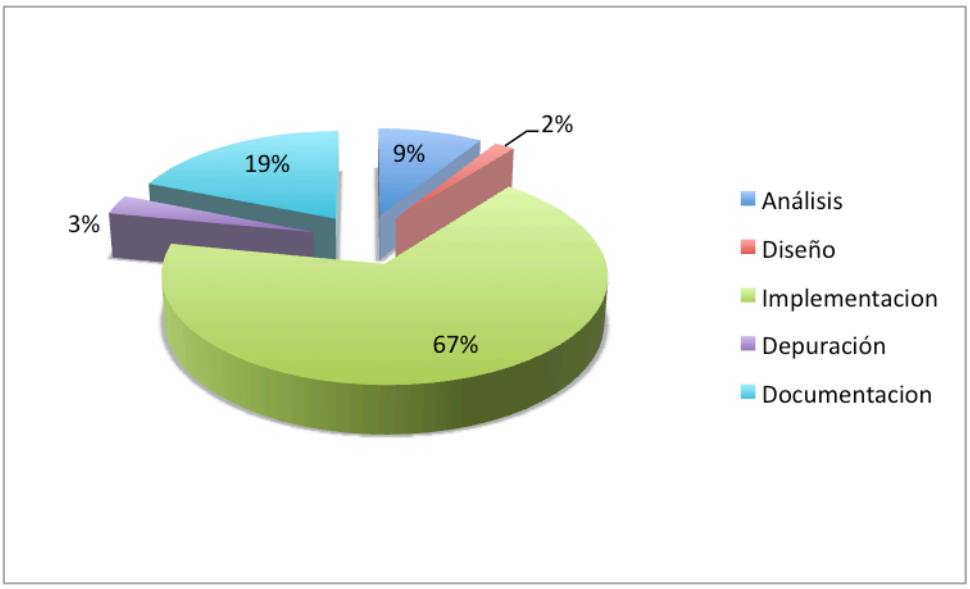

Fig 83. Grafico porcentual de las fases de l proyecto en función del tiempo

El diagrama de Gantt que muestra la planificación del mismo es el siguiente:

Fig 84. Diagrama Gantt.

8.4 Estimación de costes

Para calcular el coste estimado del proyecto se deben tener en cuenta tanto los gastos materiales como los costes humanos del desarrollo.

8.4.2 Estimación de los costes hardware.

Para el desarrollo de este proyecto se ha utilizado el siguiente equipamiento hardware:

- Ordenador personal.
    - Sistema operativo Mac OS X versión 10.6.8
    - Procesador: 2GHz Intel Core 2 Duo
    - Memoria: 2 GB 1067 MHz DDR3.
- Dispositivo Android. Samsung Galaxy
    - Pantalla táctil 4.0” WVGA (480x800) 16 M SUPER AMOLED
    - Velocidad de procesamiento de la CPU de 1 GHz.
    - Versión Android 2.1 (Éclair)
    - Almacenamiento ROM de 512 MB.
    - Almacenamiento RAM de 8GB.
    - Cámara de 5 megapíxeles.

De tal forma que el coste asociado a estos recursos es el siguiente:

Tabla 4. Costes de los recursos hardware

Recurso	Coste
Ordenador personal	950 €
Dispositivo Android	200 €
Total	1150 €

8.4.3 Estimación de los costes software

A continuación se muestra el presupuesto en forma de tabla de las herramientas software utilizadas para el desarrollo del proyecto.

Tabla 5. Costes de los recursos software

Programas	Licencia
IDE Eclipse	Gratuita
MS Office 2010	98,99 €
Mac OS X	30 €
Total	128,99 €

8.4.4 Estimación de los costes de recursos humanos

Para calcular el coste de los recursos humanos empleados durante todo el proceso se ha llevado un seguimiento diario.

El proyecto se ha realizado por:

- Un jefe de proyectos. Se encarga de la gestión del proyecto, su organización y supervisión a lo largo de todo el desarrollo del mismo.

- **Analista**. Se encarga de obtener y redactar los requisitos, además de modelar los procesos y tareas a codificar.

- **Programador**. Su tarea es desarrollar la aplicación.

- **Tester**. Su tarea es probar las distintas versiones de la aplicación, para comprobar su correcto funcionamiento en todos los casos posibles.

El tiempo empleado por los distintos profesionales es el siguiente:

	Nombre	Trabajo
1	Jefe de proyecto	206 horas
	Revisión del proyecto	*120 horas*
	Retoques del programa total	*12 horas*
	Viabilidad y estado del arte del sistema planteado	*27,5 horas*
	Informe final y manual de usuario	*39 horas*
	Seleccionar la estrategia de diseño	*7,5 horas*
2	Analista	320 horas
	Viabilidad y estado del arte del sistema planteado	*27,5 horas*
	Estudio de herramientas y alternativas de programación	*16 horas*
	Preparación de la documentación del proyecto	*175 horas*
	Listado de objetivos	*71 horas*
	Seleccionar la estrategia de diseño	*7,5 horas*
	Descomposición de los módulos de funcionalidades	*8 horas*
	Diseño de la aplicación	*8 horas*
	Aplicación de las pruebas	*7 horas*
3	Programador	370 horas
	Reajustes del código	*119 horas*
	Reajustes del código	*16 horas*
	Memoria explicativa y análisis de resultados	*24 horas*
	Implementación aplicación primera parte	*199 horas*
	Retoques del programa total	*12 horas*
4	Tester	303 horas
	Pruebas y depuración	*48 horas*
	Pruebas y depuración	*176 horas*
	Pruebas y depuración	*79 horas*

Fig 85. Utilización de los Recursos Humanos

De tal forma que el coste estimado es:

Tabla 6. Coste estimado de los recursos humanos

Recursos Humanos	Horas	Coste/Hora	Total
Jefe de proyecto	206	30	6180€
Analista	320	23	7360 €
Programador	370	20	7400 €
Tester	303	18	5454 €
Subtotal			26394 €
I.V.A (18%)			4750,92 €
Total			31144,92 €

8.5 Estimación del coste total del proyecto

El presupuesto total de este proyecto asciende a la cantidad de TREINTA Y DOS MIL DOSCIENTOS CUARENTA Y TRES EUROS CON NOVENTA CÉNTIMOS DE EURO.

Tabla 7. Coste total del proyecto

Concepto	Cantidad
Recursos hardware	1150 €
Recursos software	128,99 €
Recursos humanos	31144,92 €
Total	32243,9 €

Capítulo 9. Conclusiones y trabajo futuro

En este capítulo se hace un repaso general al proyecto, presentando las conclusiones finales y las posibles líneas de trabajos futuros.

9.1 Conclusiones finales

Una vez finalizadas las principales tareas de este Proyecto Fin de Carrera, se puede hacer balance y crítica de los resultados obtenidos repasando los objetivos iniciales:

- **Realizar un estudio de la plataforma Android**: estudio general del sistema operativo de Google.

  A lo largo de todo el proyecto se ha conseguido tener un conocimiento amplio de este nuevo Sistema Operativo: su arquitectura, sus componentes y características así como su funcionamiento y las posibilidades que ofrece. Este conocimiento ha sido posible gracias a la extensa documentación y facilidades que Google ha puesto a disposición de los desarrolladores. También gracias a foros, blogs y demás publicaciones de Internet, ya que la popularización de este Sistema Operativo ha ido creciendo desde su aparición.

  Al trabajar con Android, puede descubrirse cuáles son las principales ventajas que ofrece frente a sus competidores en el campo del desarrollo y que justifican su elección como plataforma:

- El hecho de utilizar un lenguaje tan popular como JAVA hace que cualquier programador mínimamente experimentado pueda desarrollar su aplicación sin mayores complicaciones ni necesidad de formarse o asimilar una sintaxis diferente. Incluye además las API más populares de este lenguaje como java.util, java.net o java.io

- La fácil distribución de aplicaciones en el Market hace que todo el mundo pueda estudiar, modificar y distribuir su aplicación, a la vez que permite el desarrollo privado mediante la publicación comercial de aplicaciones. Cada desarrollador puede decidir cómo distribuir su propio trabajo.

- La construcción de interfaces de usuario es un aspecto muy cuidado en Android. El desarrollador dispone de una gran cantidad de elementos y diseños que puede combinar de forma visual, o bien definirlas en código fuente mediante XML.

- **Realizar un estudio de los métodos de extracción de información a través de los descriptores más comunes en el tratamiento de imágenes**.

Para abordar este proyecto fue necesario estudiar no sólo los principios básicos del procesamiento de imágenes sino también las funciones de la librería OpenCV. Esto aporta conocimientos hasta entonces no adquiridos sobre cómo tratar las imágenes y cómo poder modificar sus características.

La necesidad de extraer las características de las imágenes para trabajar con ellas, ha llevado a estudiar dos de los descriptores más populares. Con lo que se consigue entender su funcionamiento y analizarlos para poder decidir cuál es la mejor opción a utilizar.

- **Estudio de la posibilidad de utilizar la librería OpenCV**, librería en código C, en una aplicación Android.

La necesidad de utilizar esta librería, ha hecho que se consiga tener un mayor conocimientos del uso de librerías en proyectos de desarrollo software y en la configuración del proyecto para poder importar librerías.

Cabe destacar que este punto supuso una de las mayores dificultades iniciales, puesto que son necesarias ciertas herramientas para la compilación que no funcionaban de la misma manera en Sistemas Operativos.

- **Estudio de los conceptos básicos de Teoría de Grafos y técnicas de Máximo Cliqué.**

Para abordar la segunda parte del proyecto, fue necesario estudiar conceptos de Teoría de Grafos, hasta hora no tratados con tanta profundidad y que han permitido tener un mayor conocimiento de cuál es su funcionamiento y cómo pueden ser de utilidad en la vida cotidiana.

- **Estudio de la posibilidad de aplicar Teoría de Grafos** para el matching entre dos imágenes a través de las características extraídas utilizando el descriptor seleccionado.

Este punto aporta el conocimiento de cómo es de aplicable todo lo estudiado anteriormente, de cómo es posible tratar la información extraída de una imagen y relacionarla mediante grafos y algún criterio de comparación (distancia euclídea).

- **Análisis y desarrollo de la aplicación en Android**: estudio del diseño a implementar. Estudio de las funciones del programa, sus casos de uso y la interfaz gráfica. Desarrollo de la aplicación sobre la plataforma Android.

Después de haber cumplido los objetivos anteriores, conocer las características de Android, su API y el entorno de desarrollo existente, se procedió a desarrollar una aplicación que cumpliera con los objetivos del proyecto.

Todo el diseño e implementación ha sido documentado convenientemente con el fin de ayudar al lector a comprender el funcionamiento y construcción de aplicaciones en Android utilizando funciones de procesamiento de imágenes basadas en OpenCV y Teoría de Grafos.

- **Estudio de la capacidad de respuesta del terminal y de los resultados obtenidos en función de las imágenes procesadas.**

Una vez desarrollada la aplicación el objetivo era analizar los tiempos de respuesta en un terminal de gama media.

Según los resultados mostrados en el capítulo 7 se llega a la conclusión de que el tiempo de respuesta en el procesamiento de imágenes aumenta con el tamaño de las imágenes, de tal forma que para conseguir unos resultados aceptables es necesario situarlo por debajo de 1 Megapíxel.

En cuanto al cálculo del tiempo requerido para calcular el grafo, una vez obtenidas las características, es despreciable (del orden de milisegundos), por lo que el principal aspecto a considerar es el tiempo requerido para extraer las características SURF, tiempo que crece con el tamaño de la imagen, de tal forma que para un tamaño superior a 1 Megapíxel el procesamiento llega a tardar más de 10 segundos. Por esta razón podemos concluir que, para tamaños de imágenes nos superiores a 1 Megapíxel y, más concretamente, si no superan los 0.4 Megapíxeles, el tiempo de respuesta no llega a alanzar el segundo, lo cual hace que la alternativa de realizar el *matching* entre imágenes utilizando Teoría de Grafos sea una opción a considerar en el procesamiento de imágenes.

Una vez que se han revisado los objetivos iniciales se puede concluir que todos han sido satisfechos, ya que:

- Se ha logrado desarrollar por completo la aplicación con los requisitos iniciales mediante herramientas de software libre, con lo que se consigue que sea accesible y modificable por terceros.

- Se ha conocido de forma más o menos amplia un nuevo Sistema Operativos con una gran influencia en el mercado actual.

- Se han analizado los tiempos de respuesta en un terminal de gama media, obteniendo unos resultados aceptables.

9.2 Trabajos futuros

Como líneas futuras de trabajo, se proponen los siguientes puntos:

- Optimizar la aplicación, mediante la exportación de métodos a código nativo.
- Optimizar el cálculo del grafo descartando puntos característicos no válidos.
- Realizar en *stictching* entre imágenes utilizando la información obtenida tras el *matching.*
- Estudiar la viabilidad de aplicar Teoría de Grafos a otros métodos de procesamiento de imáenes.
- Estudiar el stictching entre imágenes utilizando otras técnicas, como la información obtenida usando el sensor de movimiento.
- Ofrecer la posibilidad de guardar en formato imagen los resultados obtenidos.

Capítulo 10. Referencias

[1] Proyecto CVCamera. Último acceso 20 de octubre de 2011. http://code.google.com/p/android-opencv/wiki/CVCamera

[2] OpenCV. Último acceso 20 de octubre de 2011. http://opencv.willowgarage.com/wiki/

[3] Aplicación 360 pano. Último acceso 20 de octubre de 2011.

http://www.cs.cornell.edu/courses/cs4670/2010fa/projects/final/results/group_of_acc269_ty244_yc563/cs4670_final.html

[4] RobotView. Último acceso 20 de octubre de 2011.

http://theveganrobot.com/

[5] Proyecto librería OpenCV para Android.Google. Último acceso 1 de noviembre de 2010.

http://code.google.com/p/android-opencv

[6] Proyecto librería OpenCV para Android de Bill McCord. Último acceso 1 de noviembre de 2010.

http://billmccord.github.com/OpenCV-Android/

[7] Proyecto librería OpenCV universidad de Standford. Último acceso 1 de noviembre de 2010.

http://www.stanford.edu/~zxwang/android_opencv.html

[8] Aplicación Detección de Coches + HTC Magic+ Android+ OpenCV. Último acceso 1 de diciembre de 2010.

http://www.youtube.com/watch?v=imDz2aFMvQc

[9] Detector facial. Último acceso 20 de noviembre de 2011. http://www.youtube.com/watch?v=lLTW2xXIPKE&feature=player_embedded

[10] Word Lens. Último acceso 20 de noviembre de 2011. http://itunes.apple.com/es/app/word-lens/id383463868?mt=8

[11] Layar. Último acceso 20 de noviembre de 2011.

http://www.layar.com/

[12] Wikipedia, entrada OpenCV. Último acceso 27 de noviembre de 2010

http://es.wikipedia.org/wiki/OpenCV

[13] Descripción de la arquitectura de Android, por Google. Último acceso en 28 de noviembre de 2010

http://es.youtube.com/watch?v=QBGfUs9mQYY

[14]SDK de Android. Web del proyecto Android. Último acceso en 1 de noviembre de 2010

http://code.google.com/android/download.html

[15] Guía de instalación del SDK de Android. Web del proyecto Android. Último acceso en 1 de noviembre de 2010.

http://code.google.com/android/intro/installing.html

[16] Android NDK. Último acceso 1 de noviembre de 2010.

http://developer.android.com/sdk/ndk/index.html

[17] Barrapunto, artículo publicado el 5 de diciembre de 2007, “Por qué Google desarrolla Dalvik en ver de usar Java Micro Edition?”. Último acceso en 4 diciembre de 2010.

http://softlibre.barrapunto.com/article.pl?sid=07/12/05/1357216

[18] Plugin de Eclipse para Android.

http://developer.android.com/sdk/eclipse-adt.html

[19] Incluir SDK de Android en eclipse.

http://developer.android.com/sdk/adding-components.html

[21] [W3C 2008] World Wide Web Consortium, 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008. Último acceso el 10 de junio de 2011.

http://www.w3.org/TR/2008/REC-xml-20081126

[22] Libería JAVA para Grafos. Último acceso el 10 de junio de 2011. http://www.jgrapht.org/

[23] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: a survey," Foundations and Trends in Computer Graphics and Vision, vol. 3, pp. 177–280, July 2008.

[24] K. Derpanis, "Integral image-based representations," Department of Com- puter Science and Engineering, York University, Paper, 2007.

[25] P. Viola and M. Jones, "Robust real-time object detection," International Journal of Computer Vision, vol. 57, no. 2, pp. 137–154, 2002.

[26] Samsung Galaxy, especificaciones. Último acceso 27 de julio de 2011.

http://www.samsung.com/global/microsite/galaxys/specification/spec.html?ver=high

[27] Descarga de distintos paquetes de Eclipse. Web oficial de la plataforma Eclipse.

Último acceso en 1 de noviembre de 2010.

http://www.eclipse.org/downloads/

[28] Página oficia de HTC Wildfire. Último acceso, 1 diciembre de 2010.

http://www.htc.com/es/product/wildfire/overview.html

[29] Android Market. Requisitos para la distribución de apliaciones. Último acceso, 25 de diciembre de 2011.

http://support.google.com/androidmarket/developer/bin/answer.py?hl=es&answer=113469

[30] Reconocimiento y registro 3D de objetos conocidos en una escena. Último acceso, 25 de diciembre de 2011.

http://zaguan.unizar.es/TAZ/CPS/2010/5078/TAZ-PFC-2010-171.pdf

[31] Reconocimiento de patrones. CENATAV. Último acceso 25 de diciembre de 2011.

http://www.cenatav.co.cu/doc/RTecnicos/RT%20SerieAzul_030web.pdf

Anexo I. Android y OpenCV

Android

Android es un sistema operativo inicialmente desarrollado por Android Inc., firma comprada por Google en 2005. Está basado en una versión modificada del kernel de Linux. Es partícipe de la Open Handset Alliance, alianza comercial de 78 compañías para desarrollar estándares abiertos para dispositivos móviles.

Desde su lanzamiento ha supuesto un cambio en la forma de ver los terminales móviles y todas las posibilidades que nos podía ofrecer, permitiéndole obtener una gran cuota de mercado, motivado no sólo por todo aquello que ofrece a los usuarios sino también a los desarrolladores.

Según recientes estudios mercado, las unidades de teléfonos inteligentes con Android se ubican en el primer puesto en los Estados Unidos, en el segundo y tercer cuarto del 2010, con una cuota de mercado del 43.6% en el tercer trimestre [12 (16)].

La expansión de terminales Android y la capacidad de desarrollo de aplicaciones para este entorno gracias a que sea una plataforma de código abierto, incrementa día a día su amplia comunidad de desarrolladores los cuales permiten extender la funcionalidad de los dispositivos. A la fecha existen más de 100000 aplicaciones disponibles para Android.

Desde sus inicios en 2005 ya ha lanzado distintas versiones del sistema operativo y de la herramienta de desarrollo de aplicaciones SDK, permitiéndonos explotar cada vez más las posibilidades del terminal.

El lanzamiento oficial de Android Software Development Kit a apareció en noviembre de 2007 y a mediados de agosto de 2008 apareció Android 0.9 SDK versión beta. Al mes siguiente se produjo el lanzamiento de Android 1.0 SDK (Release 1).

A principios de marzo de 2009 Google presentó la versión 1.1 de Android para el "dev phone" y la actualización incluía algunos cambios estéticos menores además de soporte para "búsqueda por voz" y distintos arreglos en Gmail, reloj y demás.

A medidos de mayo de 2009, Google lanza la versión 1.5 de Android OS (Cupcake) con su respectivo SDK que incluía nuevas funcionalidades como: grabación de vídeo, soporte para Bluetooth, sistema de teclado personalizable, reconocimiento de voz y el AppWidget Framework que permitía los desarrolladores crear sus propios widget para la pantalla principal.

Android 1.5 fue la versión que más personas utilizaron para iniciarse en Android.

Tras esta, apareció en septiembre de 2009 Android 1.6 "Donut" con mejoras en las búsquedas, indicador de uso de batería y hasta el VPN Control Applet. Esta versión fue tan buena que todos los Android que no tienen una interfaz personalizada como HTC Sense o Motoblur corren en esta versión.

Para llevar las cosas más allá Motorola Droid fue lanzado con Android 2.0 "Eclair" que incluía nuevas funcionalidades y hasta aplicaciones precargadas que requería un hardware mucho más rápido que la generación anterior de terminales Android.

Poco después, el Google Nexus One (el cual marcó un antes y un después ya que Google trató de venderlo por su cuenta y liberado, además de en algunas operadoras), llegó con Android 2.1 con nuevas capacidades 3D, live wallpapers y lo que significó una mejora de la plataforma desde 1.6.

La última versión comercializada de este sistema operativo es Android 2.2 "Froyo", la cual incluye nuevas características como una mejora en la seguridad, mejora del rendimiento del navegador y en la gestión de memoria; y mejoras para desarrolladores como Android Cloud, informes de errores de aplicaciones, aplicaciones de almacenamiento externo, nuevas APIS y mejoras en el tratamiento de la cámara.

Si miramos al futuro de la plataforma, por un lado Android Market es la tienda de aplicaciones que más crece llegando a las 100000 aplicaciones, Android es el sistema

operativo que más está creciendo en Estados Unidos. Sin embargo levanta quejas por la fragmentación de la plataforma debido a las distintas versiones pero lo cierto es que ya se está empezando a desarrollar el know how para brindar las actualizaciones a los usuarios 2.1 y en el futuro a las siguientes.

La plataforma Android.

Android constituye una pila de software pensada especialmente para dispositivos móviles y que incluye tanto un sistema operativo, como middleware y diversas aplicaciones de usuario.

Todas las aplicaciones para Android son programadas en lenguaje Java y son ejecutadas en una máquina virtual especialmente diseñada para esta plataforma, denominada Dalvik.

A los desarrolladores se les proporciona de forma gratuita un SDK y la opción de plug-in para el entorno de desarrollo Eclipse.

Con Android se buscar reunir en una misma plataforma todos los elementos necesarios que permitan al desarrollador controlar y aprovechar al máximo cualquier funcionalidad ofrecida por un dispositivo móvil, permitiendo a su vez la integración de los dispositivos móviles en el las posibilidades ofrecidas por la Web.

Mejorar el desarrollo y enriquecer la experiencia del usuario se convierte, por tanto, en la gran filosofía de Android y en su principal objetivo.

Arquitectura

Como ya se ha mencionado, Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye, el sistema operativo, middleware y

aplicaciones básicas para el usuario. Su diseño cuenta, entre otras, con las siguientes características:

- Busca el desarrollo rápido de aplicaciones que, sean reutilizables y portables entre diferentes dispositivos.
- Los componentes básicos de las aplicaciones se pueden sustituir fácilmente por otros.
- Cuenta con su propia máquina virtual, Dalvik, que interpreta y ejecuta código escrito en Java.
- Permite la representación de gráficos 2D y 3D.
- Posibilita el uso de bases de datos.
- Soporta un elevado número de formatos multimedia.
- Servicio de localización GSM.
- Controla los diferentes elementos hardware: Bluetooth, Wi-Fi, cámara fotográfica o de vídeo, GPS, acelerómetro, etc., siempre y cuando esté disponible en el terminal.
- Cuenta con un entorno de desarrollo cuidado mediante un SDK disponible de forma gratuita.
- Ofrece plug-in para los entornos de desarrollos más populares, Eclipse y un emulador integrado para ejecutar aplicaciones.

Un visión global por capas de la arquitectura empleada en Android es la siguiente [13(6)]:

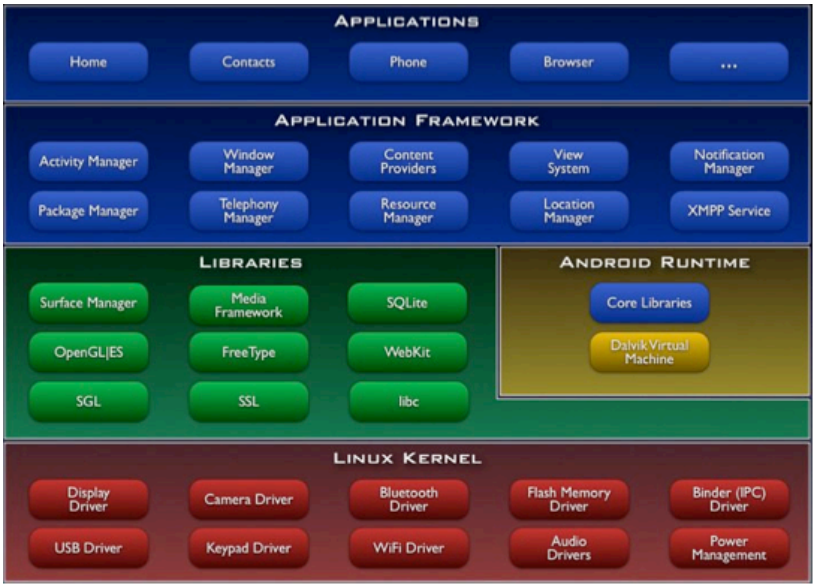

Fig. 86. Arquitectura Android.

La capa más inmediata es la que corresponde al núcleo de Android, la cual contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un nuevo fabricante incluya un nuevo elemento hardware, lo primero que se debe hacer para poder ser utilizado en Android es incluir las librerías de control o drivers necesarios en este kernel de Linux embebido en el propio Android.

La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades. Junto al núcleo basado en Linux constituyen el corazón de Android.

Al mimo nivel que las librerías de Android se encuentra el entorno de ejecución. Éste lo constituyen las Core Libraries, que son librerías con multitud de clases Java, y la máquina virtual Dalvik.

Los dos últimos niveles de la arquitectura de Android están escritos enteramente en Java. El Framework de las aplicaciones representa fundamentalmente el conjunto de las herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o por terceras compañías o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo framework, representado por este nivel.

El último nivel del diseño arquitectónico de Android son las aplicaciones. Éste nivel incluye tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente. Todas estas aplicaciones utilizan servicios, las API y librerías de los niveles anteriores.

Android SDK

Como ya hemos mencionado anteriormente el entorno de programación de aplicaciones para Android es lo que se llama " Android SDK". Consta de un paquete de librerías de desarrollo y de un emulador para pruebas y tutoriales básicos. Además es posible la instalación de un plug-in para el entorno de desarrollo Eclipse que facilita la programación y compilación del software. El lenguaje de programación empleado es Java, junto con las librerías propias de Android. El sistema operativo utiliza la máquina virtual "Dalvik", que optimiza el uso de memoria y favorece el asilamiento de procesos.

Existen hasta el momento tres versiones estables de Android SDK: versión 1.0, versión 1.1_r1 y versión 1.5, de reciente aparición.

Toda la información de esta herramienta de desarrollo, así como la descarga de la misma es posible obtenerla de forma gratuita de la página oficial de Android [14(9)][15(11)].

Android NDK

Android NDK (Native Development Kit) es una herramienta complementaria para el SDK de Android que permite a los desarrolladores trabajar con código nativo, pudiendo crear librerías que lo incluyan. Esto sólo es posible con el uso de SDK, por lo que es imprescindible que esté instalado.

NDK permite el desarrollo de aplicaciones que interactúan mucho más estrecha con el núcleo de los móviles, puesto que están escritas en código nativo.

El NDK permite aplicar en las aplicaciones a desarrollar código de lenguajes nativos como C o C++. Esto puede ser útil para ciertas aplicaciones para la reutilización de código existente, pudiendo producir un aumento de la velocidad o simplemente la utilización de librerías en C, como es nuestro caso.

NDK dispone de lo siguiente:

- Un conjunto de herramientas y crear archivos que se usan para generar códigos de librerías nativas de C y C++.

- Una manera de integrar las bibliotecas nativas en un archivo de tipo .apk que se puede ejecutar en los dispositivos Android.

- Un conjunto de cabeceras de sistema operativo nativo y de bibliotecas que contarán con el apoyo de las futuras versiones de la plataforma Android.

- Documentación, ejemplos y tutoriales.

En la página oficial de Android Developers se encuentra toda la documentación relacionada con los requisitos que debe tener el sistema, así como los pasos que se deben seguir para la instalación de dicho complemento [16(19)].

OpenCV

Una vez descrito el entorno de desarrollo que se emplea en este proyecto y las herramientas que nos ofrece de cara al desarrollo, el siguiente punto para completar la descripción del contexto que envuelve al desarrollo de la aplicación, es cómo enfrentarnos al objetivo del proyecto, tratamiento de imágenes. En esta sección se describirá la librería que nos permitirá implementar la técnica de visión artificial que persigue esta aplicación y cómo es posible tratar con ella en nuestra plataforma

OpenCV (Open Source Computer Vision Library), conjunto de bibliotecas en C y C++ de código libre (“Open Source BSD”) orientadas a visión por computador. Posee

más de 500 funciones, que incluye funciones en lenguaje nativo para aplicaciones de Visión Artificial. Originariamente desarrollada por Intel y utilizada en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea libremente usada para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Es una librería multiplataforma, versiones para Linux, Mac OS X y Windows.

OpenCV es la biblioteca idónea para el tratamiento de imágenes, ya que se diseñó para el procesamiento de imágenes, cada función y estructura de los datos se diseñó con el codificador de procesamiento de imágenes en mente, lo que supone una mayor rapidez para ejecutar acciones de este tipo. El hecho de que esté desarrollada en C/C++ supone una mayor rapidez en la ejecución de las operaciones, que comparado con otros lenguajes y herramientas con las que poder realizar tales acciones supone un mejor rendimiento. Todo esto y el hecho del gran desarrollo de esta librería, la han convertido en una librería fundamental para el procesamiento de imágenes.

Las bibliotecas que la componen se dividen en cinco grandes grupos:

- CXCORE. Donde se encuentran las estructuras y algoritmos básicos que usan las demás funciones.
- CV. Donde están implementadas las funciones principales de procesamiento de imágenes.
- HighGUI. Incluye todo aquello relacionado con la interfaz gráfica de OpenCV y las funciones que permiten importar imágenes y vídeo.
- ML. Cuenta con algoritmos de aprendizaje, clasificadores y demás.
- CvAux. Posee funciones experimentales.

Algunas de las aplicaciones más relevantes en las que ha sido utilizada son las siguientes:

- Sistema de visión del vehículo no tripulado Stanley de la Universidad de Standford, ganador en el 2005 del Gran desafío DARPA.

- Uso en sistemas de vigilancia de vídeo.

- Programa Swistrack, herramienta de seguimiento distribuida.

Uso en Android

Para el uso de las funciones proporcionadas por la librería OpenCV es necesario disponer de la herramienta complementaria de Android para el uso de código nativo, dicha herramienta complementaria es el NDK de Android.

Esta herramienta nos permitirá la compilación de la librería y de las funciones que implementaremos en lenguaje nativo para realizar la funcionalidad que nuestra aplicación requiera, mediante las correspondientes llamadas a las funciones de la librería.

Como ya se ha mencionado en el punto anterior, para el uso de NDK debemos tener correctamente instalado SDK de Android. La instalación de NDK se detalla en el Anexo I de esta memoria.

La utilización de las funciones implementadas con lenguaje nativo requiere la compilación de las mismas, así como de las funciones de la biblioteca OpenCV que utilizaremos, creando una librería con extensión .so que será la que utilizaremos en nuestro proyecto.

Las funciones serán creadas en un archivo .c y declaradas en el correspondiente archivo de cabecera .h. Estos archivos, así como los propios de la librería, los cuales nos podremos descargar de la página oficial de la misma o bien de los distintos proyectos existentes para portar la misma a Android, deberán situarse en una carpeta denominada “jni”.

La compilación con NDK para crear la librería a utilizar en nuestro proyecto Android, exige que nuestro proyecto esté estructurado de la siguiente forma:

En el primer nivel de la carpeta de nuestro proyecto tendremos el archivo "Application.mk" y el directorio "project". El archivo contiene declaradas las rutas donde se encuentra el archivo Android.mk y el directorio anterior.

En el directorio proyect, encontraremos todos los directorios propios de un proyecto Android, en el que incluiremos el directorio "jni" con los archivos de la librería OpenCV y los que incluyen las funciones creadas basadas en éste. También se incluirá en el mismo directorio el archivo "Android.mk", con las rutas de los archivos a compilar y librería y ficheros de cabecera que se utilizarán.

La compilación con NDK, la cual se detalla en el Anexo II creará la librería con extensión .so.

Una vez creada la librería, no hará falta indicarle a nuestro proyecto la existencia de la misma mediante la ejecución de alguna acción externa a nuestro código en Java, sino que lo haremos mediante la creación de una clase en la que le indicaremos las funciones nativas a exportar y la librería de la cual queremos realizar dicha exportación.

Distintas versiones de OpenCV para Android.

Existen distintos proyectos cuya finalidad es portar y optimizar la librería OpenCV para su uso en la plataforma Android. Los proyectos más destacados son los siguientes.

Proyecto para portar la librería OpenCV a Android, de los grupos de desarrolladores de Google. [5]

Proyecto para portar y optimizar la librería OpenCV para Android de Bill McCord, perteneciente a la compañía Intuitive Automata Inc (Hong Kong) [6]. Este proyecto es el más completo de los encontrados, posee una gran cantidad de funciones ya implementadas para el tratamiento de imágenes que utilizan las funciones propias de la librería OpenCV. Es recomendable utilizar una librería completa, ya que no todas ellas incluyen todas las funciones de OpenCV.

- Universidad de Standford. Departamento de Ingeniería Eléctrica. Llevado a cabo por Zixuan Wag [7]. Esta librería es más simple que la anterior, no incluye la misma cantidad de funciones.

Cabe destacar que estas librerías no trabajan con la versión más reciente de OpenCV por lo que no incluyen todas las funciones que esta nos ofrece.

Este Proyecto Fin de Carrera parte de un Estudio Tecnológico anterior, en el que se decidió analizar estas librerías y tratar con ellas. A medida que el proyecto avanzó, se decidió realizar una segunda aplicación que utilizara la última versión estable de la librería OpenCV.

Tras probar las distintas librerías, en el Estudio Tecnológico y en el Proyecto Fin de Carrera se utilizó la de Bill McCord por ser la que disponía de las funciones que se necesitaban para la implementación de la aplicación.

Anexo II. Instalar NDK

La instalación se realizará en este caso sobre el sistema operativo Windows XP SP3.

Lo primero que hay que hacer es instalar Cygwin, para usar los entornos de desarrollo de Linux . Debemos seguir los sucesivos pasos de la instalación y cuando aparezca en pantalla la lista de paquetes a instalar sólo será necesario seleccionar los de desarrollo (seleccionaremos “Install” para el paquete con la etiqueta “Devel”) .

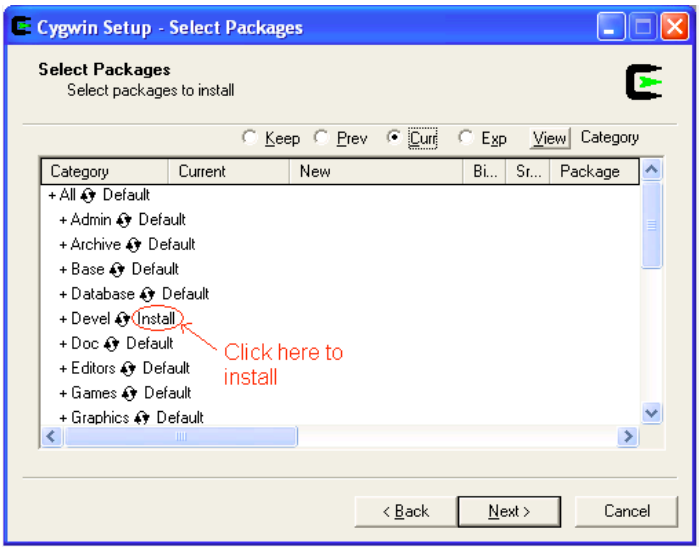

Fig. 87. Pantalla instalación Cygwin

Una vez instalado Cygwin, hay que descargar el paquete NDK Crystax para Windows [20] y lo guardaremos en el directorio “home” de Cygwin.

Tras esto se debe editar la configuración de las variables de entorno del Sistema. En Windows XP la encontraremos siguiendo la siguiente ruta: Panel de Control → Sistema →Opciones Avanzadas →Variables de entorno.

Editar la variable de entorno PATH, incluyendo al final:

C:\cygwin\usr\bin; C:\cygwin\home\Administrador\android-ndk-r4-crystax

Donde se puede ver que se añadde tanto la ruta de la aplicación Cygwin como la del paquete NDK descargado y que posteriormente trasladamos al directorio “home” de Cygwin.

Una vez que se han realizado los pasos anteriores estamos listos para compilar la librería y utilizarla en nuestro proyecto de Android.

Anexo III. Compilación de librería .so

Una vez instalada la herramienta NDK se puede proceder a la compilación de la librería que utilizaremos en nuestro proyecto de Android. Para ello, seguiremos los pasos que se describen a continuación.

En primer lugar, hay que trasladar el proyecto del que deseamos del que deseamos compilar los archivos que constituirían la librería al directorio "home" de Cygwin o en caso de no querer copiarlo a dicho directorio también tendremos la opción de crear un enlace al mismo.

La ejecución de los comandos para la compilación se hará a través de la consola Cygwin, la cual la podremos encontrar en la lista de aplicaciones instaladas en el Sistema.

En la consola de Cygwin hay que situarse en el directorio "jni" del proyecto a compilar:

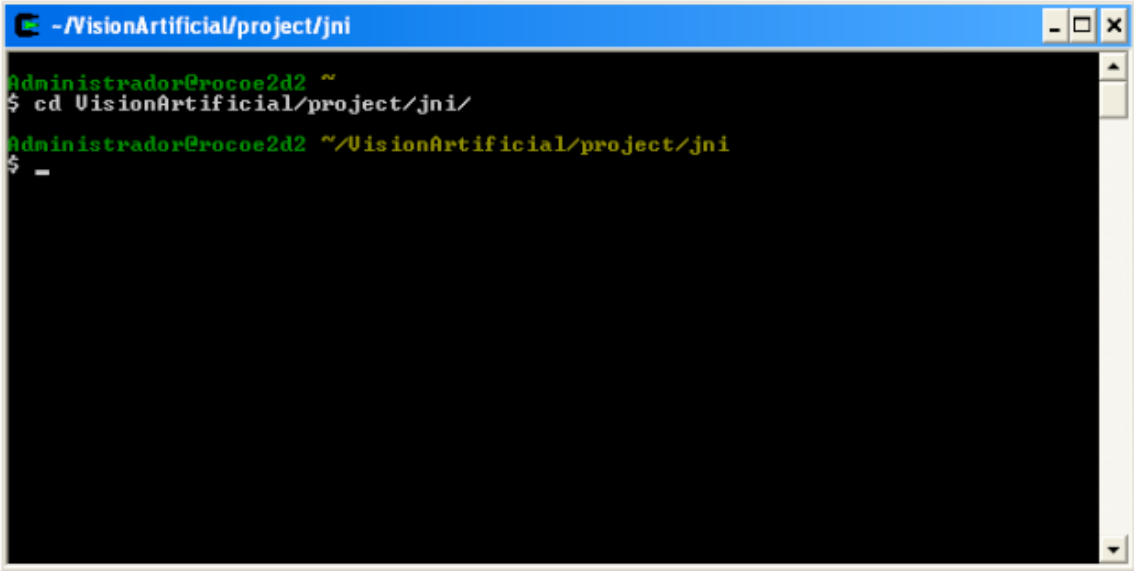

Fig. 88. Compilación librería I

y ejecutar ./ndk-build. (en el caso de seguir los pasos anteriores se corresponde al comando: ~/androide-ndk-4r-crystax/ndk-build):

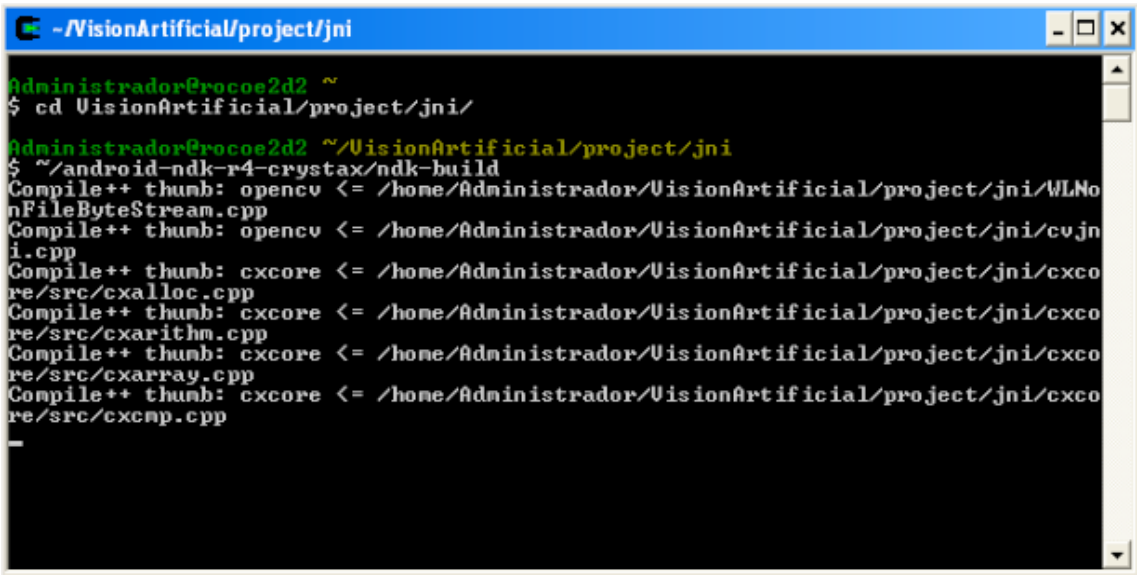

Fig. 89. Compilación librería II

Esta operación puede tardar varios minutos si es la primera vez que la ejecutamos o si en nuestro proyecto no tenemos los archivos objetos de los archivos a compilar que se crean cuando se ejecuta esta operación. Estos archivos se crearán en la carpeta "obj", mientras que la librería se creará en el directorio "libs", tal y como se nos indica al final de la compilación:

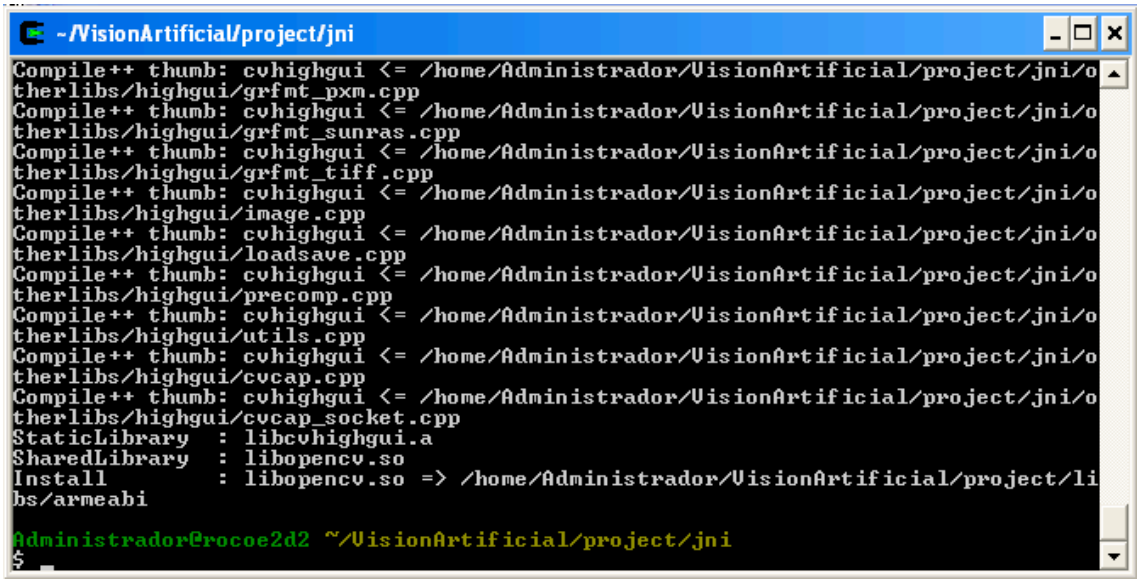

Fig. 90. Compilación librería III

Si no es la primera vez que la ejecutamos tan sólo se volverá a compilar los archivos que incluyen las modificaciones introducidas, siendo la operación mucho más rápida.